Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

ATAM:[SM] Method for Architecture Evaluation

CMU/SEI-2000-TR-004
ESC-TR-2000-004

Rick Kazman
Mark Klein
Paul Clements

August 2000

Product Line Systems

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton, Lt Col., USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

	Abstract	**vii**
1	**Introduction**	**1**
	1.1 What is the Purpose of the ATAM?	2
2	**The Underlying Concepts**	**5**
3	**A Brief Introduction to the ATAM**	**7**
4	**Quality Attribute Characterizations**	**9**
5	**Scenarios**	**13**
	5.1 Types of Scenarios	13
	5.2 Eliciting and Prioritizing Scenarios	16
	5.3 Utility Trees	16
	5.4 Scenario Brainstorming	18
6	**Attribute-Based Architectural Styles**	**19**
7	**Outputs of the ATAM**	**21**
	7.1 Risks and Non-Risks	21
	7.2 Sensitivity and Tradeoff Points	22
	7.3 A Structure for Reasoning	23
	7.4 Producing ATAM's Outputs	23
8	**The Steps of the ATAM**	**25**
	8.1 Step 1 - Present the ATAM	25
	8.2 Step 2 - Present Business Drivers	26
	8.3 Step 3 - Present Architecture	27
	8.4 Step 4 - Identify Architecture Approaches	29
	8.5 Step 5 - Generate Quality Attribute Utility Tree	29

8.6 Step 6 - Analyze Architecture Approaches 29
8.7 Step 7 - Brainstorm and Prioritize Scenarios 33
8.8 Step 8 - Analyze Architecture Approaches 36
8.9 Step 9 - Present Results 37

9 The Two Phases of ATAM 39
9.1 Phase 1 Activities 39
9.2 Phase 2 Activities 40
9.3 ATAM Steps and Their Associated Stakeholders 41
9.4 A Typical ATAM Agenda 42

10 A Sample Evaluation: The BCS 45
10.1 Phase 1 45
10.2 Phase 2 46

11 Results Of The BCS ATAM 59
11.1 Documentation 59
11.2 Requirements 59
11.3 Architectural Risks 60
11.4 Reporting Back and Mitigation Strategies 60

12 Conclusions 63

Bibliography 65

Appendix A Attribute Characteristics 67
A.1 Performance 67
A.2 Modifiability 68
A.3 Availability 69

List of Figures

Figure 1	Example Performance Related Questions	11
Figure 2	Example Attribute-Specific Questions	12
Figure 3	A Sample Utility Tree	17
Figure 4	Concept Interactions	24
Figure 5	Example Template for the Business Case Presentation	26
Figure 6	Example Template for the Architecture Presentation	28
Figure 7	Architectural Approach Documentation Template	31
Figure 8	Example Architectural Approach Description	32
Figure 9	Example Scenarios with Rankings	35
Figure 10	Highly Ranked Scenarios with Quality Attribute Annotations	36
Figure 11	A Sample ATAM Agenda	43
Figure 12	Hardware View of the BCS	47
Figure 13	A Portion of the BMS Utility Tree	48
Figure 14	Performance Characterization—Stimuli	67
Figure 15	Performance Characterization—Responses	67
Figure 16	Performance Characterization—Architectural Decisions	68
Figure 17	Modifiability Characterization	68
Figure 18	Availability Characterization	69

List of Tables

Table 1	Utility Trees vs. Scenario Brainstorming	16
Table 2	ATAM Steps Associated with Stakeholder Groups	41
Table 3	Sample Scenarios for the BCS Evaluation	56

Abstract

If a software architecture is a key business asset for an organization, then architectural analysis must also be a key practice for that organization. Why? Because architectures are complex and involve many design tradeoffs. Without undertaking a formal analysis process, the organization cannot ensure that the architectural decisions made—particularly those which affect the achievement of quality attribute such as performance, availability, security, and modifiability—are advisable ones that appropriately mitigate risks. In this report, we will discuss some of the technical and organizational foundations for performing architectural analysis, and will present the Architecture Tradeoff Analysis Method[SM] (ATAM)—a technique for analyzing software architectures that we have developed and refined in practice over the past three years.

[SM] Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

1 Introduction

The purpose of this report is to describe the theory behind the Architecture Tradeoff Analysis Method[SM](ATAM) and to discuss how it works in practice. The ATAM gets its name because it not only reveals how well an architecture satisfies particular quality goals (such as performance or modifiability), but it also provides insight into how those quality goals interact with each other—how they *trade off* against each other. Such design decisions are critical; they have the most far-reaching consequences and are the most difficult to change after a system has been implemented.

When evaluating an architecture using the ATAM, the goal is to understand the consequences of architectural decisions with respect to the quality attribute requirements of the system. Why do we bother? Quite simply, an architecture is the key ingredient in a business or an organization's technological success. A system is motivated by a set of functional and quality goals. For example, if a telephone switch manufacturer is creating a new switch, that system must be able to route calls, generate tones, generate billing information and so forth. But if it is to be successful, it must do so within strict performance, availability, modifiability, and cost parameters. The architecture is the key to achieving—or failing to achieve—these goals. The ATAM is a means of determining whether these goals are achievable by the architecture as it has been conceived, *before* enormous organizational resources have been committed to it.

We have developed an architecture analysis *method* so that the analysis is repeatable. Having a structured method helps ensure that the right questions regarding an architecture will be asked *early*, during the requirements and design stages when discovered problems can be solved relatively cheaply. It guides users of the method—the stakeholders—to look for conflicts and for resolutions to these conflicts in the software architecture.

This method has also been used to analyze legacy systems. This frequently occurs when the legacy system needs to support major modifications, integration with other systems, porting, or other significant upgrades. Assuming that an accurate architecture of the legacy system is available (which frequently must be acquired and verified using architecture extraction and conformance testing methods [Kazman 99]), applying the ATAM results in increased understanding of the quality attributes of the system.

[SM] Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

The ATAM draws its inspiration and techniques from three areas: the notion of architectural styles; the quality attribute analysis communities; and the Software Architecture AnalysisSAAM Method [Kazman 94], which was the predecessor to the ATAM. The ATAM is intended for analysis of an architecture with respect to its quality attributes. Although this is the ATAM's focus, there is a problem in operationalizing this focus. We (and the software engineering community in general) do not understand quality attributes well: what it means to be "open" or "interoperable" or "secure" or "high performance" changes from system to system, from stakeholder to stakeholder, and from community to community.

Efforts on cataloguing the implications of using design patterns and architectural styles contribute, frequently in an informal way, to ensuring the quality of a design [Buschmann 96]. More formal efforts also exist to ensure that quality attributes are addressed. These consist of formal analyses in areas such as performance evaluation [Klein 93], Markov modeling for availability [Iannino 94], and inspection and review methods for modifiability [Kazman 94].

But these techniques, if they are applied at all, are typically applied in isolation and their implications are considered in isolation. This is dangerous. It is dangerous because all design involves tradeoffs and if we simply optimize for a single quality attribute, we stand the chance of ignoring other attributes of importance. Even more significantly, if we do not analyze for multiple attributes, we have no way of understanding the tradeoffs made in the architecture—places where improving one attribute causes another one to be compromised.

1.1 What is the Purpose of the ATAM?

It is important to clearly state what the ATAM is and is not:

> **The purpose of the ATAM is to assess the consequences of architectural decisions in light of quality attribute requirements.**

The ATAM is meant to be a risk identification method, a means of detecting areas of potential risk within the architecture of a complex software intensive system. This has several implications:

- The ATAM can be done early in the software development life cycle.
- It can be done relatively inexpensively and quickly (because it is assessing architectural design artifacts).
- The ATAM will produce analyses commensurate with the level of detail of the architectural specification. Furthermore it need not produce detailed analyses of any measurable quality attribute of a system (such as latency or mean time to failure) to be successful. Instead, success is achieved by identifying *trends*.

This final point is crucial in understanding the goals of the ATAM; we are not attempting to precisely predict quality attribute behavior. That would be impossible at an early stage of design; one doesn't have enough information to make such a prediction. What we are interested in doing—in the spirit of a risk identification activity—is learning where an attribute of interest is affected by architectural design decisions, so that we can reason carefully about those decisions, model them more completely in subsequent analyses, and devote more of our design, analysis, and prototyping energies on such decisions.

Thus, what we aim to do in the ATAM, in addition to raising architectural awareness and improving the level of architectural documentation, is to record any *risks*, *sensitivity points*, and *tradeoff points* that we find when analyzing the architecture. Risks are architecturally important decisions that have not been made (e.g., the architecture team has not decided what scheduling discipline they will use, or has not decided whether they will use a relational or object oriented database), or decisions that have been made but whose consequences are not fully understood (e.g., the architecture team has decided to include an operating system portability layer, but are not sure what functions need to go into this layer). Sensitivity points are parameters in the architecture to which some measurable quality attribute response is highly correlated. For example, it might be determined that overall throughput in the system is highly correlated to the throughput of one particular communication channel, and availability in the system is highly correlated to the reliability of that same communication channel. A tradeoff point is found in the architecture when a parameter of an architectural construct is host to more than one sensitivity point where the measurable quality attributes are affected differently by changing that parameter. For example, if increasing the speed of the communication channel mentioned above improves throughput but reduces its reliability, then the speed of that channel is a tradeoff point.

Risks, sensitivity points, and tradeoff points are areas of potential future concern with the architecture. These areas can be made the focus of future effort in terms of prototyping, design, and analysis.

A prerequisite of an evaluation is to have a statement of quality attribute requirements and a specification of the architecture with a clear articulation of the architectural design decisions. However, it is not uncommon for quality attribute requirement specifications and architecture renderings to be vague and ambiguous. Therefore, two of the major goals of ATAM are to

- elicit and refine a precise statement of the architecture's driving quality attribute requirements
- elicit and refine a precise statement of the architectural design decisions

Given the attribute requirements and the design decisions, the third major goal of ATAM is to

- evaluate the architectural design decisions to determine if they satisfactorily address the quality requirements

2 The Underlying Concepts

The ATAM focuses on quality attribute requirements. Therefore, it is critical to have precise characterizations for each quality attribute. Quality attribute characterizations answer the following questions about each attribute:

- What are the stimuli to which the architecture must respond?
- What is the measurable or observable manifestation of the quality attribute by which its achievement is judged?
- What are the key architectural decisions that impact achieving the attribute requirement?

The notion of a *quality attribute characterization* is a key concept upon which ATAM is founded.

One of the positive consequences of using the ATAM that we have observed is a clarification and concretization of quality attribute requirements. This is achieved in part by eliciting scenarios from the stakeholders that clearly state the quality attribute requirements in terms of stimuli and responses. The process of brainstorming scenarios also fosters stakeholder communication and consensus regarding quality attribute requirements. *Scenarios* are the second key concept upon which ATAM is built.

To elicit design decisions we start by asking what architectural approaches are being used to achieve quality attribute requirements. Our goal in asking this question is to elicit the architectural approaches, styles, or patterns used that contribute to achieving a quality attribute requirement. You can think of an architectural style as a template for a coordinated set of architectural decisions aimed at satisfying some quality attribute requirements. For example, we might determine that a client/server style is used to ensure that the system can easily scale its performance so that its average-case latency is minimally impacted by an anticipated doubling of the number of users of the enterprise software system.

Once we have identified a set of architectural styles or approaches we ask a set of attribute-specific questions (for example, a set of performance questions or a set of availability questions) to further refine our knowledge of the architecture. The questions we use are suggested by the attribute characterizations. Armed with knowledge of the attribute requirements and the architectural approaches, we are able to analyze the architectural decisions.

Attribute-based architectural styles (ABASs) [Klein 99a] help with this analysis. Attribute-based architecture styles offer attribute-specific reasoning frameworks that illustrate how each architectural decision embodied by an architectural style affects the achievement of a quality attribute. For example, a modifiability ABAS would help in assessing whether a publisher/subscriber architectural style would be well-suited for a set of anticipated modifications. The third concept upon which ATAM is found is, thus, the notion of *attribute-based architectural styles.*

In the remainder of this report we will briefly introduce the ATAM, explain its foundations, discuss the steps of the ATAM in detail, and concludes with an extended example of applying the ATAM to a real system.

3 A Brief Introduction to the ATAM

The ATAM is an analysis method organized around the idea that architectural styles are the main determiners of architectural quality attributes. The method focuses on the identification of business goals which lead to quality attribute goals. Based upon the quality attribute goals, we use the ATAM to analyze how architectural styles aid in the achievement of these goals. The steps of the method are as follows:

Presentation

1. **Present the ATAM**. The method is described to the assembled stakeholders (typically customer representatives, the architect or architecture team, user representatives, maintainers, administrators, managers, testers, integrators, etc.).
2. **Present business drivers**. The project manager describes what business goals are motivating the development effort and hence what will be the primary architectural drivers (e.g., high availability or time to market or high security).
3. **Present architecture**. The architect will describe the proposed architecture, focussing on how it addresses the business drivers.

Investigation and Analysis

4. **Identify architectural approaches**. Architectural approaches are identified by the architect, but are not analyzed.
5. **Generate quality attribute utility tree**. The quality factors that comprise system "utility" (performance, availability, security, modifiability, etc.) are elicited, specified down to the level of scenarios, annotated with stimuli and responses, and prioritized.
6. **Analyze architectural approaches**. Based upon the high-priority factors identified in Step 5, the architectural approaches that address those factors are elicited and analyzed (for example, an architectural approach aimed at meeting performance goals will be subjected to a performance analysis). During this step architectural risks, sensitivity points, and tradeoff points are identified.

Testing

7. **Brainstorm and prioritize scenarios**. Based upon the exemplar scenarios generated in the utility tree step, a larger set of scenarios is elicited from the entire group of stakeholders. This set of scenarios is prioritized via a voting process involving the entire stakeholder group.

8. **Analyze architectural approaches**. This step reiterates step 6, but here the highly ranked scenarios from Step 7 are considered to be test cases for the analysis of the architectural approaches determined thus far. These test case scenarios may uncover additional architectural approaches, risks, sensitivity points, and tradeoff points which are then documented.

Reporting

9. **Present results**. Based upon the information collected in the ATAM (styles, scenarios, attribute-specific questions, the utility tree, risks, sensitivity points, tradeoffs) the ATAM team presents the findings to the assembled stakeholders and potentially writes a report detailing this information along with any proposed mitigation strategies.

4 Quality Attribute Characterizations

Evaluating an architectural design against quality attribute requirements necessitates a precise characterization of the quality attributes of concern. For example, understanding an architecture from the point of view of modifiability requires an understanding of how to measure or observe modifiability and an understanding of how various types of architectural decisions impact this measure. To use the wealth of knowledge that *already* exists in the various quality attribute communities, we have created characterizations for the quality attributes of performance, modifiability, and availability, and are working on characterizations for usability and security. These characterizations serve as starting points, which can be fleshed out further in preparation for or while conducting an ATAM.

Each quality attribute characterization is divided into three categories: *external stimuli, architectural decisions,* and *responses. External stimuli* (or just *stimuli* for short) are the events that cause the architecture to respond or change. To analyze an architecture for adherence to quality requirements, those requirements need to be expressed in terms that are concrete and measurable or observable. These measurable/observable quantities are described in the *responses* section of the attribute characterization. *architectural decisions* are those aspects of an architecture—components, connectors, and their properties—that have a direct impact on achieving attribute responses.

For example, the external stimuli for performance are events such as messages, interrupts, or user keystrokes that result in computation being initiated. Performance architectural decisions include processor and network arbitration mechanisms; concurrency structures including processes, threads, and processors; and properties including process priorities and execution times. Responses are characterized by measurable quantities such as latency and throughput.

For modifiability, the external stimuli are change requests to the system's software. architectural decisions include encapsulation and indirection mechanisms, and the response is measured in terms of the number of affected components, connectors, and interfaces and the amount of effort involved in changing these affected elements. Characterizations for performance, availability, and modifiability are given in Appendix A.

Our goal in presenting these attribute characterizations is not to claim that we have created an *exhaustive* taxonomy for each of the attributes, but rather to suggest a framework for thinking about quality attributes; a framework that we have found facilitates a reasoned and efficient inquiry to elicit the appropriate attribute-related information.

The attribute characterizations help to ensure attribute coverage as well as offering a rationale for asking elicitation questions. For example, irrespective of the style being analyzed we know that latency (a measure of response) is at least a function of

- resources such as CPUs and LANs
- resource arbitration such as scheduling policy
- resource consumption such as CPU execution time
- and external events such message arrivals

We know that these architectural resources must be designed so that they can ensure the appropriate response to a stimulus. Therefore, given a scenario such as "*Unlock all of the car doors within one second of pressing the correct key sequence*," the performance characterization inspires questions such as

- Is the one-second deadline a hard deadline (response)?
- What are the consequences of not meeting the one-second requirement (response)?
- What components are involved in responding to the event that initiates unlocking the door (architectural decisions)?
- What are the execution times of those components (architectural decisions)?
- Do the components reside on the same or different processors (architectural decisions)?
- What happens if several "unlock the door" events occur quickly in succession (stimuli)?

Are the servers single- or multi-threaded?	What is the location of firewalls and their impact on performance?
How are priorities assigned to processes?	What information is cached versus regenerated? Based upon what principles?
How are processes allocated to hardware?	What is the performance impact of a thin versus a thick client?
What is the physical location of the hardware and its connectivity?	How are resources allocated to service requests?
What are the bandwidth characteristics of the network?	How do we characterize client loading, (e.g., how many concurrent sessions, how many users)?
How is queuing and prioritization done in the network?	What are the performance characteristics of the middleware: load balancing, monitoring, reconfiguring services to resources?
Do you use a synchronous or an asynchronous protocol?	
What is the impact of uni-cast or multi-cast broadcast protocols?	

Figure 1: Examples of Performance Related Questions

These questions are inspired by the attribute characterizations and result from applying the characterization to architecture being evaluated. For example, see the performance questions in Figure 1 and consider how these questions might have been inspired by the performance characterization in Figure 16 of Appendix A.

Other examples of attribute-specific questions are shown in Figure 2.

Modifiability:

If this architecture includes layers/facades, are there any places there where the layers/facades are circumvented?

If this architecture includes a data repository, how many distinct locations in the architecture have direct knowledge of its data types and layout?

If a shared data type changes, how many parts of the architecture are affected?

Performance:

If there are multiple processes competing for a shared resource, how are priorities assigned to these processes and the process controlling the resource?

If there are multiple pipelines of processes/threads, what is the lowest priority for each process/thread in each pipeline?

If multiple message streams arrive at a shared message queue, what are the rates and distributions of each stream?

Are there any relatively slow communication channels along an important communication path (e.g., a modem)?

Availability:

If redundancy is used in the architecture, what type of redundancy (analytic, exact, functional) and how is the choice made between redundant components?

How are failures identified?

Can active as well as passive failures be identified?

If redundancy is used in the architecture, how long does it take to switch between instances of a redundant component?

Figure 2: Examples of Attribute-Specific Questions

5 Scenarios

In a perfect world, the quality requirements for a system would be completely and unambiguously specified in a requirements document that is evolving ahead of or in concert with the architecture specification. In reality, requirements documents are not written, or are written poorly, or do not properly address quality attributes. In particular, we have found that quality attribute requirements for both existing and planned systems are missing, vague, or incomplete. Typically the first job of an architecture analysis is to precisely elicit the specific quality goals against which the architecture will be judged. The mechanism that we use for this elicitation is the *scenario*.

A scenario is a short statement describing an interaction of one of the stakeholders with the system. A *user* would describe using the system to perform some task; his scenarios would very much resemble *use cases* in object-oriented parlance. A *maintainer* would describe making a change to the system, such as upgrading the operating system in a particular way or adding a specific new function. A *developer*'s scenario might talk about using the architecture to build the system or predict its performance. A *customer*'s scenario might describe how the architecture is to be re-used for a second product in a product line.

Scenarios provide a vehicle for concretizing vague development-time qualities such as modifiability; they represent specific examples of current and future uses of a system. Scenarios are also useful in understanding run-time qualities such as performance or availability. This is because scenarios specify the kinds of operations over which performance needs to be measured, or the kinds of failures the system will have to withstand.

5.1 Types of Scenarios

In ATAM we use three types of scenarios: *use case scenarios* (these involve typical uses of the existing system and are used for information elicitation); *growth scenarios* (these cover anticipated changes to the system), and *exploratory scenarios* (these cover extreme changes that are expected to "stress" the system). These different types of scenarios are used to probe a system from different angles, optimizing the chances of surfacing architectural decisions at risk. Examples of each type of scenario follow.

5.1.1 Use Case Scenarios

Use case scenarios describe a user's intended interaction with the completed, running system. For example,

1. There is a radical course adjustment during weapon release (e.g., loft) that the software computes in 100 ms. (performance)
2. The user wants to examine budgetary and actual data under different fiscal years without re-entering project data. (usability)
3. A data exception occurs and the system notifies a defined list of recipients by e-mail and displays the offending conditions in red on data screens. (reliability)
4. User changes graph layout from horizontal to vertical and graph is redrawn in one second. (performance)
5. Remote user requests a database report via the Web during peak period and receives it within five seconds. (performance)
6. The caching system will be switched to another processor when its processor fails, and will do so within one second. (reliability)

Notice that each of the above use case scenarios expresses a specific stakeholder's desires. Also, the stimulus and the response associated with the attribute are easily identifiable. For example, in the first scenario, "radical course adjustment during weapon release," the stimulus and latency goal of 100 ms. is called out as being the important response measure. For scenarios to be well-formed it must be clear what the stimulus is, what the environmental conditions are, and what the measurable or observable manifestation of the response is.

Notice also that the attribute characterizations suggest questions that can be helpful in refining scenarios. Again consider Scenario 1:

- Are data sampling rates increased during radical course adjustments?
- Are real-time deadlines shortened during radical course adjustments?
- Is more execution time required to calculate a weapon solution during radical course adjustments?

5.1.2 Growth Scenarios

Growth scenarios represent typical anticipated future changes to a system. Each scenario also has attribute-related ramifications, many of which are for attributes other than modifiability. For example, Scenarios 1 and 4 will have performance consequences and Scenario 5 might have performance, security and reliability implications.

1. Change the heads-up display to track several targets simultaneously without affecting latency.
2. Add a new message type to the system's repertoire in less that a person-week of work.
3. Add a collaborative planning capability where two planners at different sites collaborate on a plan in less than a person-year of work.
4. The maximum number of tracks to be handled by the system doubles and the maximum latency of track data to the screen is kept to 200 ms.
5. Migrate to a new operating system, or a new release of the existing operating system in less than a person-year of work.
6. Add a new data server to reduce latency in use case Scenario 5 to 2.5 seconds within one person-week.
7. Double the size of existing database tables while maintaining 1 second average retrieval time.

5.1.3 Exploratory Scenarios

Exploratory scenarios push the envelope and stress the system. The goal of these scenarios is to expose the limits or boundary conditions of the current design, exposing possibly implicit assumptions. Systems are never conceived to handle these kinds of modifications, but at some point in the future these might be realistic requirements for change. And so the stakeholders might like to understand the ramifications of such changes. For example,

1. Add a new 3-D map feature, and a virtual reality interface for viewing the maps in less than five person-months of effort.
2. Change the underlying Unix platform to a Macintosh.
3. Re-use the 25-year-old software on a new generation of the aircraft.
4. The time budget for displaying changed track data is reduced by a factor of 10.
5. Improve the system's availability from 98% to 99.999%.
6. Half of the servers go down during normal operation without affecting overall system availability.
7. Tenfold increase in the number of bids processed hourly while keeping worst-case response time below 10 seconds.

5.2 Eliciting and Prioritizing Scenarios

Scenarios are elicited and prioritized in ATAM using different mechanisms at different times with different stakeholder participation. The two mechanisms employed are utility trees and structured brainstorming. Table 1 highlights their common differences.

	Utility Trees	Facilitated Brainstorming
Stakeholders	Architects, project leader	All stakeholders
Typical Group size	2 evaluators; 2-3 project personnel	4-5 evaluators; 5-10 project-related personnel
Primary Goals	Elicit, concretize and prioritize the driving quality attribute requirements. Provide a focus for the remainder of the evaluation.	Foster stakeholder communication to validate quality attribute goals elicited via the utility tree.
Approach	Top-down (general to specific)	Bottom-up (specific to general)

Table 1: Utility Trees vs. Scenario Brainstorming

5.3 Utility Trees

Utility trees provide a top-down mechanism for directly and efficiently translating the business drivers of a system into concrete quality attribute scenarios. For example, in an e-commerce system two of the business drivers might be stated as: "security is central to the success of the system since ensuring the privacy of our customers' data is of utmost importance"; and "modifiability is central to the success of system since we need to be able to respond quickly to a rapidly evolving and very competitive marketplace." Before we can assess the architecture, these system goals must be made more specific and more concrete. Moreover, we need to understand the relative importance of these goals versus other quality attribute goals, such as performance, to determine where we should focus our attention during the architecture evaluation. Utility trees help to concretize and prioritize quality goals. An example of a utility tree is shown in Figure 3.

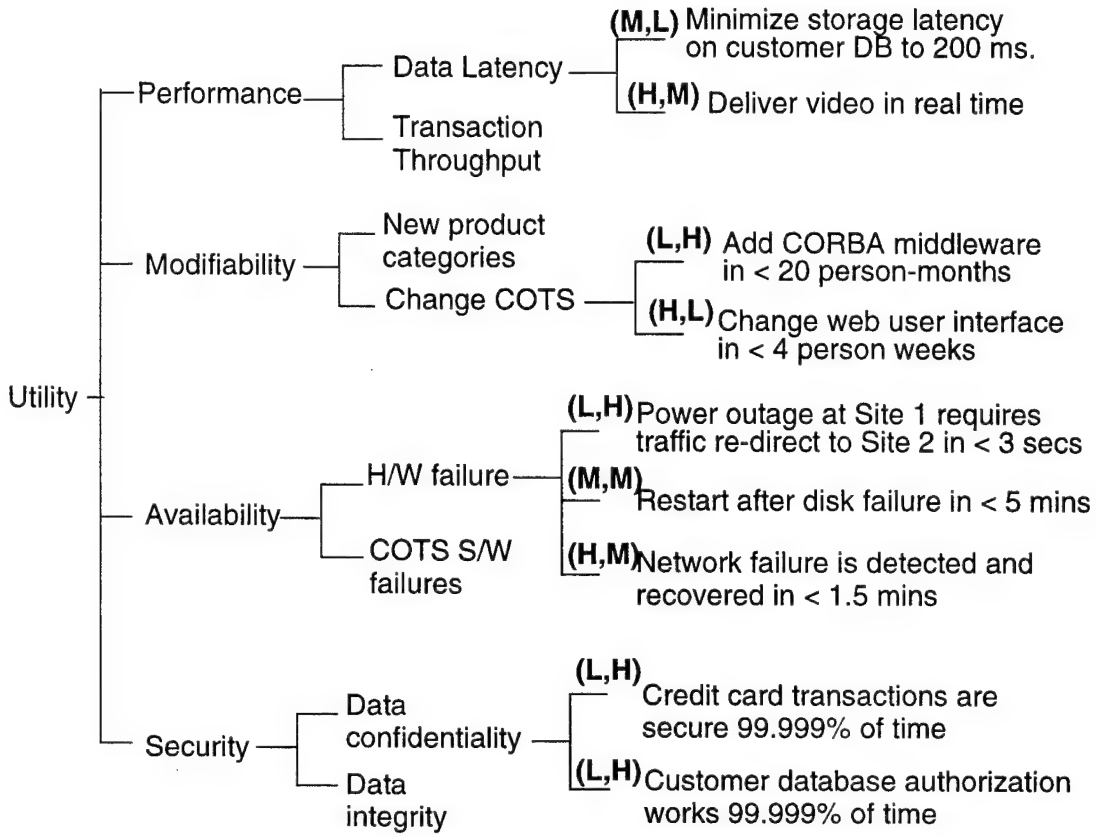

Figure 3: A Sample Utility Tree

The utility tree shown in Figure 3 contains *utility* as the root node. This is an expression of the overall "goodness" of the system. Typically the quality attributes of performance, modifiability, security, and availability are the high-level nodes immediately under utility, although different stakeholder groups may add their own quality attributes or may use different names for the same ideas (for example, some stakeholders prefer to speak of maintainability). Under each of these quality factors are specific sub-factors. For example, performance is broken down into "data latency" and "transaction throughput". This is a step toward refining the attribute goals to be concrete enough for prioritization. Notice how these sub-factors are related to the attribute characterizations. Latency and throughput are two of the types of response measures noted in the attribute characterization. Data latency is then further refined into "Minimize storage latency on customer database" and "Deliver video in real-time." Throughput might be refined into "Maximize average throughput to the authentication server." Further work on these scenarios would, in fact, make these architectural response goals even more specific, e.g., "Storage latency on customer database no more than 10 ms. on average."

These leaf nodes are now specific enough to be prioritized relative to each other. This prioritization may be on a 0-1 scale, or using relative rankings such as High, Medium, and Low; we typically prefer the latter approach as we find that the stakeholders cannot reliably and repeatably make finer distinctions than High, Medium, and Low. The prioritization of the utility tree is done along two dimensions: by the importance of each node to the success of the system and the degree of perceived risk posed by the achievement of this node (i.e., how easy the architecture teams feel this level of performance, modifiability, or other attribute will be to achieve). For example, "Minimize storage latency on customer database" has priorities of (M,L), meaning that it is of medium importance to the success of the system and low risk to achieve, while "Deliver video in real time" has priorities of (H,M), meaning that it is highly important to the success of the system and the achievement of this scenario is perceived to be of medium risk.

Refining the utility tree often leads to interesting and unexpected results. For example, in the example given in Figure 3, security and modifiability were initially designated by the stakeholders as the key attributes driving quality requirements. The subsequent elicitation and refinement of the quality attribute requirements via the utility tree resulted in determining that performance and availability were also important. Simply stated, creating the utility tree guides the key stakeholders in considering, explicitly stating, and prioritizing all of the current and future driving forces on the architecture.

The output of utility tree generation provides a prioritized list of scenarios that serves as a plan for the remainder of the ATAM. It tells the ATAM team where to spend its (relatively limited) time, and in particular where to probe for architectural approaches and risks. The utility tree guides the evaluators to look at the architectural approaches involved with satisfying the high priority scenarios at the leaves of the utility tree. Additionally, the utility tree serves to concretize the quality attribute requirements, forcing the evaluation team and the customer to define their "XYZ-ility" requirements very precisely. Statements, commonly found in requirements documents, such as "The architecture shall be modifiable and robust" are untenable here, because they have no operational meaning: they are not refutable.

5.4 Scenario Brainstorming

While utility tree generation is primarily used to understand how the architect perceived and handled quality attribute architectural drivers, the purpose of scenario brainstorming is to take the "pulse" of the larger stakeholder community. Scenario brainstorming works well in larger groups, creating an atmosphere in which the ideas and thoughts of one person stimulate others to think. The process fosters communication, creativity, and serves to express the collective mind of the participants. The prioritized list of brainstormed scenarios is compared with those generated via the utility tree exercise. If they agree, great. If additional driving scenarios are discovered, this is also an important outcome.

6 Attribute-Based Architectural Styles

An architectural style (as defined by Shaw and Garlan [Shaw 96] and elaborated on by others [Buschmann 96]) includes a description of component types and their topology, a description of the pattern of data and control interaction among the components, and an informal description of the benefits and drawbacks of using that style. Architectural styles are important since they differentiate classes of designs by offering experiential evidence of how each class has been used along with qualitative reasoning to explain why each class has certain properties."Use pipes and filters when reuse is desired and performance is not a top priority" is an example of the type of description that is a portion of the definition of the pipe and filter style.

A style can be thought of as a set of constraints on an architecture—constraints on component types and their interactions—and these constraints define the set or family of architectures that satisfy them. By locating architectural styles in an architecture, we see what strategies the architect has used to respond to the system's driving performance goals, modifiability goals, availability goals, and so forth. The ATAM uses a particular specialization of this called attribute-based architectural styles, or ABASs [Klein 99a, 99b].

An ABAS is an architectural style in which the constraints focus on component types and patterns of interaction that are particularly relevant to quality attributes such as performance, modifiability, security, or availability. ABASs aid architecture evaluation by focusing the stakeholders' attention on the patterns that dominate the architecture. This focus is accomplished by suggesting attribute-specific questions associated with the style. The attribute-specific questions are, in turn, inspired by the attribute characterizations that we discussed above. For example, a performance ABAS highlights architectural decisions that are relevant to performance—how processes are allocated to processors, where they share resources, how their priorities are assigned, and so forth.

A particular performance ABAS will highlight a subset of those questions. For example, if an architecture uses a collection of interacting processes to achieve a processing goal within a specified time period, this would be recognized as a performance ABAS. The questions associated with this performance ABAS would probe important architectural decisions such as the priority of the processes, estimates of their execution time, places where they synchronize, queuing disciplines, etc.; information that is relevant to understanding the performance of this style. The answers to these questions then feed into an explicit analytic model such as RMA (rate monotonic analysis) [Klein 93] or queuing analysis for performance. By associating ana-

lytic models with architectural styles in ABASs, we can probe the strengths and weaknesses of the architecture with respect to each quality attribute.

Examples of ABASs include

- Modifiability Layering ABAS: analyzes the modifiability of the layers architectural style by examining potential effects of various modification scenarios.
- Performance Concurrent pipelines ABAS: applies rate monotonic analysis to multiples pipelines on a single processor, each of which have real-time deadlines.
- Reliability Tri-modular redundancy ABAS: applies Markov modeling to a classical style of redundancy used to enhance system reliability.

ABASs have been described elsewhere (e.g., Klein) and will not be discussed in depth here [Klein 99a, 99b]. The point in introducing them is that ABASs have proven to be useful in architecture evaluations even if none of the catalogued styles have been explicitly identified in the architecture. All architectures have embedded patterns, since no large system is a random collection of components and connection. But the patterns might not always be apparent; they might not always be documented; they might not be "pure." And the patterns might not always be desirable. But nevertheless there are patterns. Calling out these patterns during an architecture evaluation and explicitly reasoning about their behavior by asking attribute-specific questions and applying attribute-specific reasoning models is central to the evaluation. This is how we understand the "forest" of an architecture, without getting mired in understanding individual "trees."

7 Outputs of the ATAM

While the clarification of requirements and better architecture specifications are a positive consequence of performing an architecture evaluation, the central goal of an architecture evaluation is to uncover key architectural decisions. In particular, we want to find key decisions that pose risks for meeting quality requirements and key decisions that have not yet been made. Finally, the ATAM helps an organization to develop a set of analyses, rationale, and guidelines for ongoing decision making about the architecture. This framework should live and evolve as the system evolves.

7.1 Risks and Non-Risks

Risks are potentially problematic architectural decisions. Non-risks are good decisions that rely on assumptions that are frequently *implicit* in the architecture. Both should be understood and explicitly recorded.[1]

The documenting of risks and non-risks consist of

- an architectural decision (or a decision that has not been made)
- a specific quality attribute response that is being addressed by that decision along with the consequences of the predicted level of the response
- a rationale for the positive or negative effect that decision has on meeting the quality attribute requirement

An example of a risk is

> The rules for writing business logic modules in the second tier of your three-tier client-server style are not clearly articulated (*a decision that has not been made*). This could result in replication of functionality thereby compromising modifiability of the third tier (*a quality attribute response and its consequences*). Unarticulated rules for writing the business logic can result in unintended and undesired coupling of components (*rationale for the negative effect*).

1. Risks can also emerge from other sources. For example, having a management structure that is misaligned with the architectural structure might present an organizational risk. Insufficient communication between the stakeholder groups and the architect is a common kind of management risk.

An example of a non-risk is

> Assuming message arrival rates of once per second, a processing time of less than 30 ms, and the existence of one higher priority process (*the architectural decisions*), a one-second soft deadline seems reasonable (*the quality attribute response and its consequences*) since the arrival rate is bounded and the preemptive effects of higher priority processes is known and can be accommodated (*the rationale*).

Note that for a non-risk to remain a non-risk the assumptions must not change (or at least if they change, the designation of non-risk will need to be re-justified). For example, if the message arrival rate, the processing time or the number of higher priority processes changes, the designation of non-risk could change.

7.2 Sensitivity and Tradeoff Points

We term key architectural decisions *sensitivity points* and *tradeoff points*. A sensitivity point is a property of one or more components (and/or component relationships) that is critical for achieving a particular quality attribute response. For example:

- The level of confidentiality in a virtual private network might be sensitive to the number of bits of encryption.
- The latency for processing an important message might be sensitive to the priority of the lowest priority process involved in handling the message.
- The average number of person days of effort it takes to maintain a system might be sensitive to the degree of encapsulation of its communication protocols and file formats.

Sensitivity points tell a designer or analyst where to focus attention when trying to understand the achievement of a quality goal. They serve as yellow flags: "Use caution when changing this property of the architecture." Particular values of sensitivity points may become risks when realized in an architecture. Consider the examples above. A particular value in the encryption level—say, 32 bit encryption—may present a risk in the architecture. Or having a very low priority process in a pipeline that processes an important message may become a risk in the architecture.

Sensitivity points use the language of the attribute characterizations. So, when performing an ATAM, we use the attribute characterizations as a vehicle for suggesting questions and analyses that guide us to potential sensitivity points. For example, the priority of a specific component (an architectural parameter) might be a sensitivity point if it is a key property for achieving an important latency goal (a response) of the system. A method interface (an archi-

tectural parameter) might be a sensitivity point if the amount of work required to change the interface (a response) is key in achieving a certain class of important system extensions.

A *tradeoff point* is a property that affects more than one attribute and is a sensitivity point for more than one attribute. For example, changing the level of encryption could have a significant impact on both security and performance. Increasing the level of encryption improves the predicted security but requires more processing time. If the processing of a confidential message has a hard real-time latency requirement then the level of encryption could be a tradeoff point. Tradeoff points are the most critical decisions that one can make in an architecture, which is why we focus on them so carefully.

Finally, it is not uncommon for an architect to answer an elicitation question by saying: "we haven't made that decision yet". In this case you cannot point to a component or property in the architecture and call it out as a sensitivity point because the component or property might not exist yet. However, it is important to flag key decisions that have been made as well as key decisions that have not yet been made.

7.3 A Structure for Reasoning

We do not do extensive formal modeling in an ATAM; this is not the intention of the method. The ATAM's primary goal is in determining where sensitivity points and tradeoffs exist. These areas of the architecture can then be made the objects of further in-depth analysis. The identification of sensitivity points and tradeoff points is frequently the result of *implicit* or *qualitative* analysis. That is, either from prior experience or from a basic understanding of the quality attribute in question, the evaluator is able to conclude that a quality goal is sensitive to certain properties of the architecture. A goal of any architecture evaluation is to make this reasoning explicit and to record it for posterity.

The reasoning is not always highly formal and mathematical, but it must be predictive and repeatable. The reasoning might manifest itself as a discussion that follows from the exploration the architectural approaches that address a scenario; it may be a qualitative model of attribute-specific behavior of the architecture; or it may be a quantitative model that represents how to calculate a value of a particular quality attribute. ABASs and quality attribute characterizations provide the technical foundations for creating these reasoning models.

7.4 Producing ATAM's Outputs

The power of the ATAM derives in part from the foundations provided by the technical concepts discussed here. These concepts work together synergistically.

The stimulus/response branches of the quality attribute characterizations provide a vocabulary for describing the problem the architecture is intended to solve. In Step 2 of the ATAM the customer presents the business context for the problem being solved. The utility tree is used to translate the business context first into quality attribute drivers and then into concrete scenarios that represent each business driver. Each scenario is described in terms of a stimulus and the desired response. These scenarios are prioritized in terms of how important they are to the overall mission of the system and the perceived risk in realizing them in the system. The highest priority scenarios will be used in the analysis of the architecture.

In Step 3 the system architect presents the architecture to the evaluation team. The architect is encouraged to present the architecture in terms of the architectural approaches that are used to realize the most important quality attribute goals. The evaluation team also searches for architectural styles and approaches as the architecture is presented. The approaches that address the highest priority scenarios will be the subject of analysis during the remainder of the evaluation. Attribute-specific models (such as queuing models, modifiability models, and reliability models) as they apply to specific architectural styles, are codified in ABASs. These models provide attribute-specific questions that the evaluators employ to elicit the approaches used by the architect to achieve the quality attribute requirements.

The three components used to find risks, sensitivity points, and tradeoff points are shown in Figure 4: high-priority scenarios, attribute-specific questions (as guided and framed by the attribute characterizations), and architectural approaches. During an evaluation the architect traces the scenarios through the subset of the architecture represented by the approaches. To do this the architect must identify the components and connectors involved in realizing the scenario. As the architect identifies the relevant components the evaluators ask attribute-specific questions. Some of the answers to these questions lead to the identification of sensitivity points, some of which will turn out to be risks and/or tradeoff points.

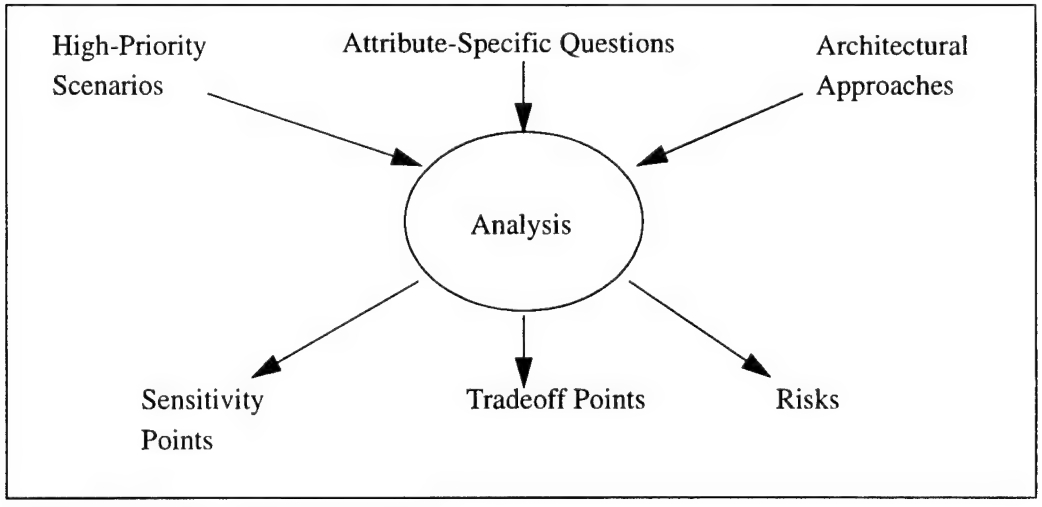

Figure 4: Concept Interactions

8 The Steps of the ATAM

The ATAM process consists of nine steps, as was briefly presented in Section 3. Sometimes there must be dynamic modifications to the order of steps to accommodate the availability of personnel or architectural information. Although the steps are numbered, suggesting linearity, this is not a strict waterfall process. There will be times when an analyst will return briefly to an earlier step, or will jump forward to a later step, or will iterate among steps, as the need dictates. The importance of the steps is to clearly delineate the activities involved in ATAM along with the outputs of these activities.

The amount of time it takes to carry out an ATAM varies depending on the size of the system and the maturity of the architecture and the state of its description. We have conducted ATAMs by executing the steps in three consecutive days. However we have found that the most effective ATAMs have been those in which the ATAM team and the customer establish a common understanding of the architecture during an initial phase of the evaluation and then later assemble a larger group of stakeholders for a more formal evaluation meeting.

8.1 Step 1 - Present the ATAM

In this step the evaluation team lead presents the ATAM to the assembled stakeholders. This time is used to explain the process that everyone will be following, allows time to answer questions, and sets the context and expectations for the remainder of the activities. It is important that everyone knows what information will be collected, how it will be scrutinized, and to whom it will be reported. In particular, the presentation will describe

- the ATAM steps in brief
- the techniques that will be used for elicitation and analysis: utility tree generation, architectural approach-based elicitation/analysis, and scenario brainstorming/mapping
- the outputs from the evaluation: the scenarios elicited and prioritized, the questions used to understand/evaluate the architecture, a "utility" tree, describing and prioritizing the driving architectural requirements, the set of identified architectural approaches and styles, the set of risks and non-risks discovered, the set of sensitivity points and tradeoffs discovered

8.2 Step 2 - Present Business Drivers

The system to be evaluated needs to be understood by all participants in the evaluation. In this step the project manager presents a system overview from a business perspective. A sample outline for such a presentation is given in Figure 5. The system itself must be presented, initially at a high level of abstraction, typically describing

- its most important functional requirements
- its technical, managerial, economic, or political constraints
- its business goals and context
- its major stakeholders
- the architectural drivers (major quality attribute goals that shape the architecture)

8.2.1 Business Case/Architecture Presentation

To ensure the quality, consistency, and volume of information presented in Steps 2 and 3 (the business driver and architecture presentations) we typically provide documentation templates to the presenters. An example of the documentation templates that we provide for the business case is shown in Figure 5.

Business Context/Drivers Presentation (~ 12 slides; 45 minutes) - Description of the business environment, history, market differentiators, driving requirements, stakeholders, current need and how the proposed system will meet those needs/ requirements (3-4 slides) - Description of business constraints (e.g., time to market, customer demands, standards, cost, etc.) (1-3 slides) - Description of the technical constraints (e.g., COTS, interoperation with other systems, required hardware or software platform, reuse of legacy code, etc.) (1-3 slides) - Quality attributes desired (e.g., performance, availability, security, modifiability, interoperability, integrability) and what business needs these are derived from (2-3 slides) - Glossary (1 slide)

Figure 5: Example of Template for the Business Case Presentation

8.3 Step 3 - Present Architecture

The architecture will be presented by the lead architect (or architecture team) at an appropriate level of detail. What is an appropriate level? This depends on several factors: how much information has been decided upon and documented; how much time is available; how much risk the system faces. This is an important step as the amount of architectural information available and documented will directly affect the analysis that is possible and the quality of this analysis. Frequently the evaluation team will need to specify additional architectural information that is required to be collected and documented before a more substantial analysis is possible.

In this presentation the architecture should cover

- technical constraints such as an OS, hardware, or middleware prescribed for use
- other systems with which the system must interact
- architectural approaches used to meet quality attribute requirements

At this time the evaluation team begins its initial probing of architectural approaches.

An example of the template for the architecture presentation is shown in Figure 6. Providing these templates to the appropriate stakeholders (in this case, the architect) well in advance of the ATAM helps to reduce the variability of the information that they present and also helps to ensure that we stay on schedule.

Architecture Presentation (~20 slides; 60 minutes)

- Driving architectural requirements (e.g., performance, availability, security, modifiability, interoperability, integrability), the measurable quantities you associate with these requirements, and any existing standards/models/approaches for meeting these (2-3 slides)

- High Level Architectural Views (4-8 slides)

+ functional: functions, key system abstractions, domain elements along with their dependencies, data flow

+ module/layer/subsystem: the subsystems, layers, modules that describe the system's decomposition of functionality, along with the objects, procedures, functions that populate these and the relations among them (e.g., procedure call, method invocation, callback, containment).

+ process/thread: processes, threads along with the synchronization, data flow, and events that connect them

+ hardware: CPUs, storage, external devices/sensors along with the networks and communication devices that connect them

- architectural approaches or styles employed, including what quality attributes they address and a description of how the styles address those attributes (3-6 slides)

- Use of COTS and how this is chosen/integrated (1-2 slides)

- Trace of 1-3 of the most important use case scenarios. If possible, include the run-time resources consumed for each scenario (1-3 slides)

- Trace of 1-3 of the most important change scenarios. If possible, describe the change impact (estimated size/difficulty of the change) in terms of the changed components, connectors, or interfaces. (1-3 slides)

- Architectural issues/risks with respect to meeting the driving architectural requirements (2-3 slides)

- Glossary (1 slide)

Figure 6: Example of Template for the Architecture Presentation

8.4 Step 4 - Identify Architectural Approaches

The ATAM focuses on analyzing an architecture by understanding its architectural approaches. In this step they are identified by the architect, and captured by the analysis team, but are not analyzed.

We concentrate on identifying architectural approaches and architectural styles [1] because these represent the architecture's means of addressing the highest priority quality attributes; that is, the means of ensuring that the critical requirements are met in a predictable way [Buschmann 96, Shaw 96]. These architectural approaches define the important structures of the system and describe the ways in which the system can grow, respond to changes, withstand attacks, integrate with other systems, and so forth.

8.5 Step 5 - Generate Quality Attribute Utility Tree

In this step the evaluation team works with the architecture team, manager, and customer representatives to identify, prioritize, and refine the system's most important quality attribute goals, as described in Section 5.3. This is a crucial step in that it guides the remainder of the analysis. Analysis, even at the level of software architecture, is not inherently bound in scope. So we need a means of focussing the attention of all the stakeholders on the aspects of the architecture that are most critical to the system's success. We do this by building a utility tree.

The output of the utility tree generation process is a prioritization of specific quality attribute requirements, realized as scenarios. This prioritized list provides a guide for the remainder of the ATAM. It tells the ATAM team where to spend its limited time, and in particular where to probe for architectural approaches and their consequent risks, sensitivity points, and tradeoffs. Additionally, the utility tree serves to concretize the quality attribute requirements, forcing the evaluation team and the customer to define their "ility" requirements precisely.

8.6 Step 6 - Analyze Architectural Approaches

Once the scope of the evaluation has been set by the utility tree elicitation, the evaluation team can then probe for the architectural approaches that realize the important quality attributes. This is done with an eye to documenting these architectural decisions and identifying their risks, sensitivity points, and tradeoffs.

---

1. We look for approaches and styles because not all architects are familiar with the language of architectural styles, and so may not be able to enumerate a set of styles used in the architecture. But every architect makes architectural decisions, and the set of these we call "approaches." These can certainly be elicited from any conscientious architect.

What are we eliciting in this step? We are eliciting sufficient information about each architectural approach to conduct a rudimentary analysis about the attribute for which the approach is relevant. What are we looking for in this rudimentary analysis? We want to be convinced that the instantiation of the approach in the architecture being evaluated holds significant promise for meeting the attribute-specific requirements for which it is intended.

The major outputs of this phase are a list of architectural approaches or styles, the questions associated with them, and the architect's response to these questions. Frequently a list of risks, sensitivity points, and tradeoffs are generated. Each of these is associated with the achievement of one or more utility tree sub-factors (see Section 5.3), with respect to the quality-attribute questions that probed the risk. In effect, the utility sub-factor tells us where to probe the architecture (because this is a highly important factor for the success of the system), the architect (hopefully) responds with the architectural approach that answers this need, and we use the quality attribute-specific questions to probe the approach more deeply. The questions help us to

- understand the approach
- look for well-known weaknesses with the approach
- look for the approach's sensitivity points
- find interactions and tradeoffs with other approaches

In the end, each of these may provide the basic material for the description of a risk and this is recorded in an ever-growing list of risks.

The first action in this step is to associate the highest priority quality attribute requirements (as identified in the utility tree of Step 5) with the architectural approaches (from Step 4) used to realize them. For each highly ranked scenario generated by the utility-tree-creation step, the architect should identify the components, connectors, configuration, and constraints involved.

The evaluation team and the architecture team address each architectural approach presented by asking a set of approach-specific and quality-attribute-specific questions. These questions might come from documented experience with styles (as found in ABASs and their associated quality attribute characterizations), from books on software architecture (e.g., [Bass 98, Buchmann 96, Shaw 96, Soni 00]), or from the prior experiences of the assembled stakeholders. In practice we mine all three areas for questions.

These questions are not an end unto themselves. Each is a starting point for discussion, and for the determination of a potential risk, non-risk, sensitivity point, or tradeoff point. These, in turn, may catalyze a deeper analysis, depending on how the architect responds. For example, if the architect cannot characterize client loading and cannot say how priorities are allocated to processes and how processes are allocated to hardware, then there is little point doing a sophis-

ticated queuing or RMA performance analysis [Klein 93]. If such questions can be answered, then we perform at least a rudimentary, or back-of-the-envelope, analysis to determine if these architectural decisions are problematic or not vis-a-vis the quality attribute requirements they are meant to address. The level of analysis is not meant to be comprehensive and detailed but rather commensurate with the level of detail of the architectural specification. In other words, this will require some engineering judgement. However, the key thought to keep in mind is the need to establish some link between the architectural decisions that have been made and the quality attribute requirements that need to be satisfied. For example, it might be critical to understand how the priority of a server process responsible for managing access to shared data will impact the latency of clients who will access the shared data.

The template for capturing an architectural approach is shown in Figure 7.

Scenario: <a scenario from the utility tree of from scenario brainstorming>

Attribute: <performance, security, availability, etc.>

Environment: <relevant assumptions about the environment in which the system resides >

Stimulus: <a precise statement of the quality attribute stimulus (e.g., failure, threat, modification, ...) embodied by the scenario>

Response: <a precise statement of the quality attribute response (e.g., response time, measure of difficulty of modification)>

Architectural Decisions	**Risk**	**Sensitivity**	**Tradeoff**
<list of architectural decisions affecting quality attribute response>	<risk #>	<sens. point #>	<tradeoff #>

Reasoning:

<qualitative and/or quantitative rationale for why the list of architectural decisions contribute to meeting the quality attribute response requirement>

Architecture diagram:

<diagram or diagrams of architectural views annotated with architectural information to support the above reasoning. These may be accompanied by explanatory text.>

Figure 7: Architectural Approach Documentation Template

So, for example, we might elicit the following information from an architect, in response to a utility tree scenario that required high availability from a system, as shown in Figure 8.

Scenario: S12 (Detect and recover from HW failure of main switch.)

Attribute: Availability

Environment: normal operations

Stimulus: CPU failure

Response: 0.999999 availability of switch

Architectural decisions	**Risk**	**Sensitivity**	**Tradeoff**
Backup CPU(s)	R8	S2	
No backup Data Channel	R9	S3	T3
Watchdog		S4	
Heartbeat		S5	
Failover routing		S6	

Reasoning:

- ensures no common mode failure by using different hardware and operating system (see Risk 8)
- worst-case rollover is accomplished in 4 seconds as computing state takes ...
- guaranteed to detect failure with 2 seconds based on rates of heartbeat and watchdog ...
- watchdog is simple and proven reliable
- availability requirement might be at risk due to lack of backup data channel ... (see Risk 9)

Architecture diagram:

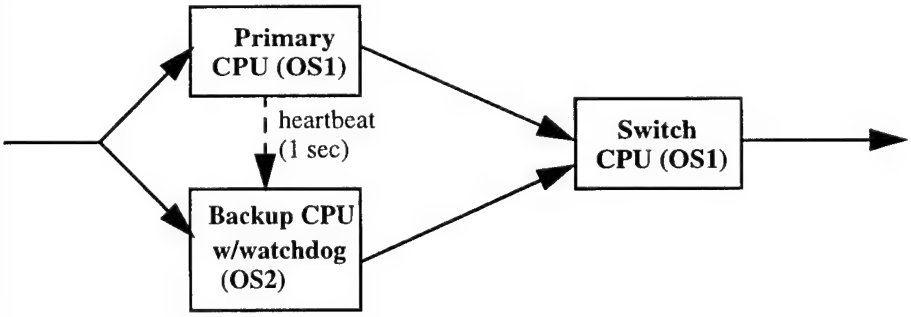

Figure 8: Example of Architectural Approach Description

As shown in Figure 8, based upon the results of this step we can identify and record a set of sensitivity points, tradeoff points, risks, and non-risks. (Recall that a *sensitivity point* is a property of one or more components and/or component relationships that is critical for achieving a

particular quality. Some of these property values will represent risks for the success of the architecture.)

All sensitivity points and tradeoff points are candidate risks. By the end of the ATAM, all sensitivity points and tradeoff points should be categorized as either a risk or a non-risk. The risks/non-risks, sensitivity points, and tradeoffs are gathered together in three separate lists. The numbers R8, T3, or S4 from Figure 8 are simply pointers into these lists.

At the end of this step, the team should now have a broad-brush picture of the most important aspects of the entire architecture, the rationale for design decisions that have been made, and a list of risks, sensitivity points, and tradeoff points. To recapitulate, let us point out where risks, sensitivity points and tradeoff points emerge. Recall that we ask attribute-specific questions of each architectural approach elicited; these questions probe the architectural approaches. The answers to these questions are potential sensitivity points. For some value of the sensitivity points we will have architectural risks and for some value we will have non-risks.

For example, the processor speed might control the number of transactions a transaction server can process in a second. Thus the processor speed is a sensitivity point with respect to the response of transactions per second. Some processor speeds will result in unacceptable values of this response—these are risks. Finally, when it turns out that an architectural decision is a sensitivity point for more than one attribute it is designated as a tradeoff point. At this point we can now test the utility and our understanding of the architectural representations that we have generated. This is the purpose of the next two steps.

8.7 Step 7 - Brainstorm and Prioritize Scenarios

Scenarios are the motor that drives the testing phase of the ATAM. Generating a set of scenarios has proven to be a great facilitator of discussion and brainstorming, when greater numbers of stakeholders are gathered to participate in the ATAM. Scenarios are examples of architectural *stimuli* used to both

- represent stakeholders' interests
- understand quality attribute requirements

The stakeholders now undertake two related activities: brainstorming *use case scenarios* (representing the ways in which the stakeholders expect the system to be used) and *change scenarios* (representing the ways in which the stakeholders expect the system to change in the future). Use case scenarios are a kind of scenario where the stakeholder is an end user, using the system to execute some function. Change scenarios represent changes to the system and are sub-divided into two categories: *growth scenarios* and *exploratory scenarios*.

Growth scenarios represent ways in which the architecture is expected to accommodate growth and change in the moderate near term: expected modifications, changes in performance or availability, porting to other platforms, integration with other software, and so forth. Exploratory scenarios, on the other hand, represent extreme forms of growth: ways in which the architecture might be stressed by changes: dramatic new performance or availability requirements (order of magnitude changes, for example), major changes in the infrastructure or mission of the system, and so forth. Growth scenarios are a way of showing the strengths and weaknesses of the architecture with respect to anticipated forces on the system. Exploratory scenarios are an attempt to find sensitivity points and tradeoff points. The identification of these points help us assess the limits of the system with respect to the models of quality attributes that we build.

Once the scenarios have been collected, they must be prioritized. We typically do this via a voting procedure where each stakeholder is allocated a number of votes equal to 30% of the number of scenarios, rounded up. So, for instance, if there were eighteen scenarios collected, each stakeholder would be given six votes. These votes can be allocated in any way that the stakeholder sees fit: all six votes allocated to one scenario, two votes to each of three scenarios, one vote to each of six scenarios, etc. In addition, at this point any stakeholder can suggest merging multiple scenarios if they are felt to represent the same stimulus/response behavior.

The prioritization and voting can be an open or a secret balloting procedure. Once the votes have been made, they are tallied and the scenarios are prioritized. A cutoff is typically made that separates the high-priority scenarios from the lower ones, and only the high-priority ones are considered in future evaluation steps. For example, a team might only consider the top five scenarios.

Figure 9 shows an example set of scenarios from a recent evaluation of a dispatching system (only the top five scenarios of the more than thirty collected are shown here), along with their votes.

4. Dynamically replan a dispatched mission within 10 minutes. [28]

27. Split the management of a set of vehicles across multiple control sites. [26]

10. Change vendor analysis tools after mission has commenced without restarting system. [23]

12. Retarget a collection of diverse vehicles to handle an emergency situation in less than 10 seconds after commands are issued. [13]

14. Change the data distribution mechanism from CORBA to a new emerging standard with less than six person-months' effort. [12]

Figure 9: Examples of Scenarios with Rankings

At this point we pause and compare the result of scenario prioritization with the results of the utility tree exercise and look for agreement or disagreement. Any difference between the high priority scenarios and the high priority nodes of the utility tree needs to be reconciled or at least explained. Reconciliation might entail clarifying the meaning of a scenario or changing one of the prioritizations. Explaining might entail understanding that the criteria used for prioritizing the utility tree were different than the criteria used in prioritizing scenarios. In any event, this is a good opportunity to make sure that all of the needs of the stakeholders are well understood and are not in irreconcilable conflict.

For example, consider Figure 10, where the highly ranked scenarios are shown along with an indication of the quality attribute or attributes that each scenario affects most heavily.

Scenario	#Votes	Quality Attributes
4	28	Performance
27	26	Performance, Modifiability, Availability
10	23	Integrability
12	13	Performance
14	12	Modifiability

Figure 10: Highly Ranked Scenarios with Quality Attribute Annotations

The utility tree generation and the scenario brainstorming activities reflect the quality attribute goals, but via different elicitation means and typically addressing different groups of stakeholders. The architects and key developers most often create the initial utility tree. The widest possible group of stakeholders is involved in the scenario generation and prioritization. Comparing the highly ranked outputs of both activities often reveals disconnects between what the architects believe to be important system qualities and what the stakeholders as a whole believe to be important. This, by itself, can reveal serious risks in the architecture, by highlighting entire areas of concern to which the architects have not attended. After this step the utility tree is the authoritative repository of the detailed high-priority quality attribute requirements from *all* sources.

8.8 Step 8 - Analyze Architectural Approaches

After the scenarios have been collected and so analyzed, the architect then begins the process of mapping the highest ranked scenarios onto whatever architectural descriptions have been presented. Ideally this activity will be dominated by the architect's mapping of scenarios onto previously discussed architectural approaches. In fact the whole point of the hiatus between the two phases is to *ensure* that this is the case. If this is not the case then either the architect has no approach- or style-based (and hence no architecture-wide) solution for the stimulus that the scenario represents, or the approach exists but was not revealed by any activities up until this point.

In this step we reiterate Step 6, mapping the highest ranked newly generated scenarios onto the architectural artifacts thus far uncovered. Assuming Step 7 didn't produce any high-priority scenarios that were not already covered by previous analysis, Step 8 is a testing activity: We hope and expect to be uncovering little new information. This is a testing activity: at this point we hope and expect to be uncovering little new information.

If we do uncover new information then this was a failing of our utility tree exercise and the architectural approaches that it led us to investigate. At this point we would need to go back to Step 4 and work through it, as well as Steps 5 and 6 until no new information is uncovered.

8.9 Step 9 - Present Results

Finally, the collected information from the ATAM needs to be summarized and presented back to the stakeholders. This presentation typically takes the form of a verbal report accompanied by slides but might, in addition, be accompanied by a more complete written report delivered subsequent to the ATAM. In this presentation we recapitulate the steps of the ATAM and all the information collected in the steps of the method including: the business context, driving requirements, constraints, and the architecture. Most important, however, is the set of ATAM outputs:

- the architectural approaches/styles documented
- the set of scenarios and their prioritization
- the set of attribute-based questions
- the utility tree
- the risks discovered
- the non-risks documented
- the sensitivity points and tradeoff points found

Each of these findings will be described and in some cases we might offer some mitigation strategies. Because we are systematically working through and trying to understand the architectural approaches it is inevitable that, at times, we make some recommendations on how the architecture might have been designed or analyzed differently. These mitigation strategies may be process related (e.g., a database administrator stakeholder should be consulted before completing the design of the system administration user interface), they may be managerial (e.g., three sub-groups within the development effort are pursuing highly similar goals and these should be merged), or they may be technical (e.g., given the estimated distribution of customer input requests, additional server threads need to be allocated to ensure that worst-case latency does not exceed five seconds). However, offering mitigation strategies is *not* an integral part of

the ATAM. The ATAM is about locating architectural risks. Addressing them may be done in any number of ways.

9 The Two Phases of ATAM

Up to this point in this report we have enumerated and described the steps of the ATAM. In this section we will describe how the steps of the ATAM are carried out over time. You will see that they are typically carried out in two phases. The first phase is architecture-centric and concentrates on eliciting architectural information and analyzing it. The second phase is stakeholder-centric and concentrates on eliciting stakeholder points of view and verifying the results of the first phase.

The reason for having two phases is that in Phase 1 we are gathering information: a subset of the evaluation team (typically one to three people), primarily interacting with the architect and one or two other key stakeholders (such as a project manager or customer/marketing representative), walks through all of the steps of the ATAM, gathering information. In almost every case this information is incomplete or inadequate to support the analysis. At this point the evaluation team interacts with the architect to elicit the necessary information. This interaction typically takes several weeks. When we feel that enough information has been collected and documented to support the goals of the evaluation, we proceed on to Phase 2.

9.1 Phase 1 Activities

The ATAM team (or more commonly a subset of the ATAM team) meets with a subset of the team being evaluated, perhaps for the first time. This meeting has two concerns: organization of the rest of the analysis activities, and information collection. Organizationally, the manager of the team being evaluated needs to make sure that the right people attend the subsequent meetings, that people are prepared, and that they come with the right attitude (i.e., a spirit of non-adversarial teamwork).

The first day is typically run as a miniature version of the entire ATAM process, concentrating on Steps 1-6. The information collection aspect of the first meeting is meant to ensure that the architecture is truly evaluable, meaning that it is represented in sufficient detail. Also, some initial "seed" scenario collection and analysis may be done on Day 1, as a way of understanding the architecture, understanding what information needs to be collected and represented, and understanding what it means to generate scenarios.

So, for example, on Day 1 the architect might present a portion of the architecture, identify some of the architectural styles or approaches, create a preliminary utility tree, and walk through a selected set of the scenarios, showing how each affects the architecture (e.g., for

modifiability) and how the architecture responds to it (e.g., for quality attributes such as performance, security and availability). Other stakeholders who are present, such as key developers, the customer, or the project manager, can contribute to describing the business context, building the utility tree, and the scenario generation processes.

Phase 1 is a small meeting, typically between a small subset of both the evaluation team and the customer team. The reason for this is that the evaluation team needs to gather as much information as possible to determine

- if the remainder of the evaluation is feasible and should proceed
- if more architectural documentation is required and, if so, precisely what kinds of documentation and how it should be represented
- what stakeholders should be present for Phase 2

At the end of this day, we will ideally have a good picture of the state and context of the project, its driving architectural requirements, and the state of its architectural documentation.

Depending upon how much is accomplished during the first phase, there is a hiatus between the first meeting and the start of the second meeting in which ongoing discovery and analysis are performed by the architecture team, in collaboration with the evaluation team. As we described earlier we do not foresee the building of detailed analytic models during this phase, but rather the building of rudimentary models that will give the evaluators and the architect sufficient insight into the architecture to make the Phase 2 meeting more productive. Also, during this hiatus the final composition of the evaluation team is determined, according to the needs of the evaluation, availability of human resources, and the schedule. For example, if the system being evaluated was safety critical, a safety expert might be recruited, or if it was database-centric then an expert in database design would be made part of the evaluation team.

9.2 Phase 2 Activities

At this point, it is assumed that the architecture has been documented in sufficient detail to support verification of the analysis already performed, and further analysis if needed. The appropriate stakeholders have been gathered and have been given advance reading materials such as a description of the ATAM, the seed scenarios, and system documentation including the architecture, business case, and key requirements. These reading materials aid in ensuring that the stakeholders know what to expect from the ATAM.

Since there will be a broader set of stakeholders attending the next meeting, and since a number of days or weeks may have transpired between the first and second meeting, it is useful to quickly recap the steps of the ATAM, so that all attendees have the same understanding and

expectations of the day's activities. In addition, it is useful to briefly recap each step immediately before it is executed.

9.3 ATAM Steps and Their Associated Stakeholders

Running an ATAM can involve as few as three to five stakeholders[1] or as many as 40 or 50. We have done evaluations at both ends of the spectrum and everywhere in between. As we have suggested, the process need not involve every stakeholder at every step. Depending on the size, criticality, and complexity of the system, the evaluation team might be smaller or larger. If the system has complex quality attribute requirements or is in a complex and unusual domain, specialists may be needed to augment the expertise of the core evaluation team. In addition, the characteristics of the system may determine which customer representatives may want to attend. Depending on whether the system is purely in-house development, shrink-wrapped software, part of a product line, or part of a industry-wide standards-based effort, different customer representatives will want to attend to ensure that their stake in the system's success is well represented. Table 2 below lists the typical categories of attendees for each ATAM step. [2]

Step	Activity	Stakeholder Groups
1	Present the ATAM	Evaluation team/Customer representatives/Architecture team
2	Present business drivers	"
3	Present architecture	"
4	Identify architectural approaches	"
5	Generate quality attribute utility tree	"
6	Analyze architectural approaches	"
7	Brainstorm and prioritize scenarios	All stakeholders
8	Analyze architectural approaches	Evaluation team/Customer representatives/Architecture team
9	Present results	All stakeholders

Table 2: ATAM Steps Associated with Stakeholder Groups

1. "All stakeholders" includes developers, maintainers, operators, reusers, end users, testers, system administrators, etc.

2. The full teams don't need to be represented for Steps 1-3, particularly in Phase 1.

9.4 A Typical ATAM Agenda

While every ATAM is slightly different than every other one, the broad brush activities do not change. In Figure 11 we show an example of a typical ATAM agenda. Each activity in this figure is followed by its step number, where appropriate, in parentheses. While we do not slavishly follow the times here, experience has shown that this represents a reasonable partitioning

of the available time in that we spend proportionately more time on those activities that experience has shown to produce more results (in terms of finding architectural risks).

Day 1

8:30	Introductions/ATAM Presentation (1)	
10:00	Customer Presents Business Drivers (2)	
10:45	Break	
11:00	Customer Presents Architecture (3)	
12:00	Identify Architecture Approaches (4)	
12:30	Lunch	
1:45	Quality Attribute Utility Tree Generation (5)	Phase 1
2:45	Analyze Architecture Approaches (6)	
3:45	Break	
4:00	Analyze Architecture Approaches (6)	
5:00	Adjourn for the Day	

Break of several weeks

Day 2

8:30	Introductions/ATAM Presentation (1)	
9:15	Customer Presents Business Context/Drivers (2)	
10:00	Break	
10:15	Customer Presents Architecture (3)	
11:15	Identify Architecture Approaches (4)	
12:00	Lunch	
1:00	Quality Attribute Utility Tree Generation (5)	
2:00	Analyze Architecture Approaches (6)	
3:30	Break	
3:45	Analyze Architecture Approaches (6)	
5:00	Adjourn for the Day	

Day 3

8:30	Introductions/Recap ATAM	
8:45	Analyze Architecture Approaches (6)	
9:30	Scenario Brainstorming (7)	
10:30	Break	
10:45	Scenario Prioritization (7)	
11:15	Analyze Architecture Approaches (8)	
12:30	Lunch	
1:30	Analyze Architecture Approaches (8)	
2:45	Prepare Report of Results/Break	
3:30	Present Results (9)	
4:00	Further Analysis/Assignment of Action Items	
5:00	Adjourn	

Phase 2

Figure 11: A Sample ATAM Agenda

10 A Sample Evaluation: The BCS

We now present an example of the ATAM as it was realized in an evaluation. Although some of the details have been changed for pedagogical reasons and to protect the identity and intellectual property of the customer and contractor, the architectural risks that we uncovered are not materially affected by these changes.

We will call this system BCS (Battlefield Control System). This system is to be used by army battalions to control the movement, strategy, and operations of troops in real time in the battlefield. This system is currently being built by a contractor, based upon government furnished requirements.

In describing the BCS ATAM, we will not describe every step of the method. This is for two reasons. The first reason is that the steps involve a reasonable amount of repetition (of activities such as scenario elicitation); in describing the method we simply describe a single instance of each activity. The second reason is that no particular instance of the ATAM slavishly follows the steps as stated. Systems to be evaluated are in different stages of development, customers have different needs, and there will be different levels of architectural documentation in different development organizations. Each of these pressures may result in *slight* changes to the schedule in practice.

10.1 Phase 1

During the initial meeting we first presented the ATAM and the customer presented the business case. The customer's business case was dominated by information on the mission and its requirements. For example, the requirements state that the system supports a Commander who commands a set of Soldiers and their equipment, including many different kinds of weapons and sensors. The system needs to interface with numerous other systems (such as command-and-control systems) that feed it commands and intelligence information. These external systems also periodically collect the BMS system's status with respect to its current missions. The business case also presented requirements with respect to a standard set of hardware and software that was to be employed, the need for extreme levels of system robustness, and a number of performance goals.

Next, the contractor presented the architecture and the contractor and customer together described their initial quality attribute requirements and their initial set of scenarios. As a result of this meeting additional architectural documentation was requested. As is often the

case in evaluating architectures, the initial documentation that was produced was far too vague and limited in scope to support any analysis. The documentation consisted of high level data flows and divisions of functionality that had no clear mapping to software constructs. There was no discussion of architectural styles or approaches in the provided documentation.

Owing to the paucity of available documented architectural information, we did not investigate any architectural approaches or map any scenarios onto the architecture. Instead we created a plan to improve the architectural documentation for use in Phase 2: we requested specific additional architectural information to address the gaps in the documentation produced by the contractor. These requests were in the form of questions such as

- What is the structure of the message-handling software (i.e., how is the functionality is broken down in terms of modules, functions, APIs, layers, etc.)?
- What facilities exist in the software architecture (if any) for self-testing and monitoring of software components?
- What facilities exist in the software architecture (if any) for redundancy, liveness monitoring, failover, and how data consistency is maintained (so that one component can take over from another and be sure that it is in a consistent state with the failed component)?
- What is the process and/or task view of the system, including mapping of these processes/tasks to hardware and the communication mechanisms between them?
- What functional dependencies exist among the software components (often called a "uses" view)?
- What data is kept in the database (which was mentioned by one of your stakeholders), how big is it, how much does it change, and who reads/writes it?
- What is the anticipated frequency and volume of data being transmitted among the system components?

Between Phases 1 and 2 of the evaluation the contractor answered many of these questions and produced substantially more complete, more analyzable architectural documentation. This documentation formed the basis for the remainder of the ATAM.

10.2 Phase 2

After recapping the steps of the ATAM to the gathered stakeholders, Phase 2 consisted of building a utility tree (using the Phase 1 utility tree as a starting point), collecting architectural approach information, collecting and prioritizing scenarios, and mapping these onto the architecture.

10.2.1 Architectural Documentation

As mentioned above, the architectural documentation covered several different views of the system: a dynamic view, showing how subsystems communicated; a set of message sequence charts, showing run-time interactions; a system view, showing how software was allocated to hardware; and a source view, showing how components and subsystems were composed of objects. For the purpose of this example, we will just show the highest level hardware structure of the architecture, using the notation from [Bass 98].[1]

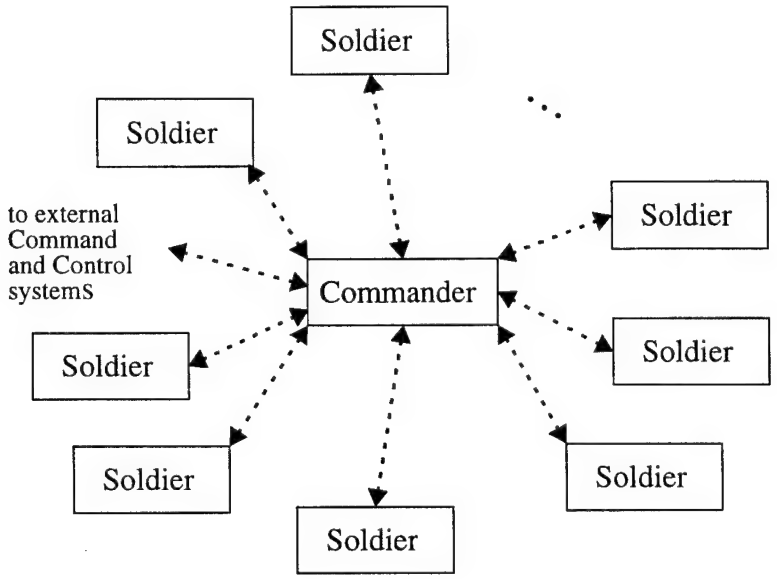

Figure 12: Hardware View of the BCS

This hardware architecture, shown in Figure 12, illustrates that the Commander is central to the system. In fact, the Commander node acts as a server and the Soldier nodes are its clients, making requests of it and updating the server's database with their status. Inter-node communication between the clients and the server is only through encrypted messages sent via a radio modem; neither subsystem controls the other. Note also that the radio modem is a shared communication channel: only one node can be broadcasting at any moment.

10.2.2 Identifying Architectural Approaches

We elicited information on the architectural approaches with respect to modifiability, availability, and performance scenarios. As stated above, the system was loosely organized around the notion of clients and servers. This dictated both the hardware architecture and the process

1. In this notation, rectangles represent processors and dashed lines represent data flow.

architecture and affected the system's performance characteristics, as we shall see. In addition to this style

- for availability, a backup commander approach was described
- for modifiability, standard subsystem organizational patterns were described
- for performance, an independent communicating components approach was described.

Each of these approaches was probed for risks, sensitivities, and tradeoffs via our style-specific questions, as we shall show below.

10.2.3 Eliciting the Utility Tree

The stakeholders in this ATAM were most interested in modifiability and performance. Upon probing, however, they admitted that availability was also of great importance to them. Based upon their stated concerns and our elicitation, a utility tree was created. A portion of this is shown in Figure 13. As part of our elicitation process we ensured that each of the scenarios in the utility tree had a specific stimulus and response associated with it.

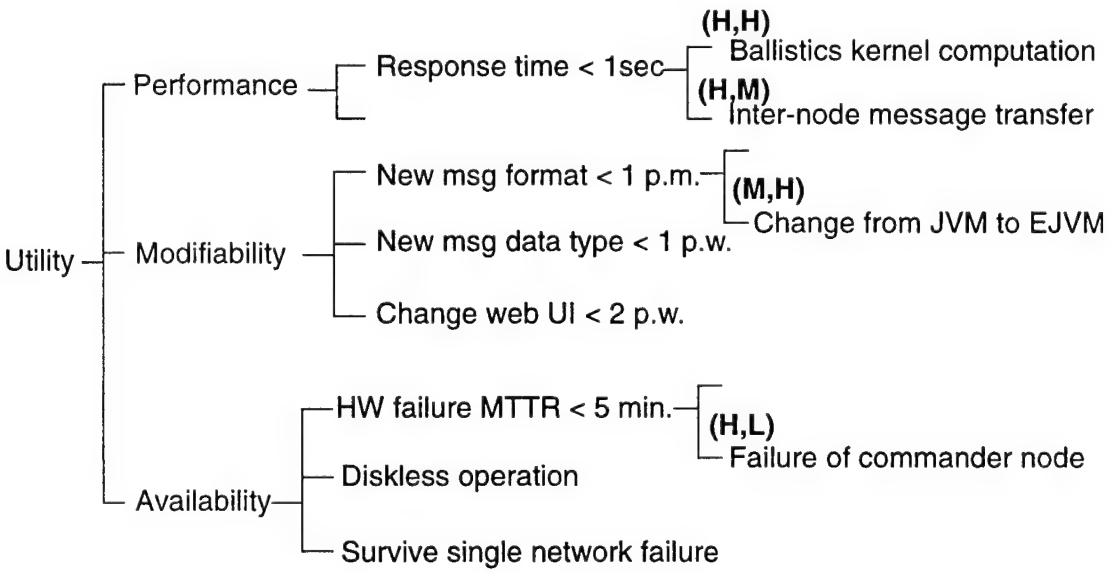

Figure 13: A Portion of the BMS Utility Tree

10.2.4 Analyze Architectural Approaches

At this stage in the analysis we had in our possession a set of architectural documentation including some architectural approaches that had been identified, but we had little insight into the way that the architectural approaches actually performed to accomplish the BCS's mission. So we used a set of screening questions and qualitative analysis heuristics to flesh out our

understanding of the approaches and to probe them for risks, sensitivity points, and tradeoffs. For example, for the backup commander approach (for availability) we generated a set of questions. The following is a subset of the questions applied to the style:

- How is the failure of a component detected?
- How is the failure of a communication channel detected?
- What is the latency for a spare component to be turned into a working replacement?
- By what means is a failed component marked for replacement?
- How are the system's working components notified that they should use the services of the spare?

These questions allowed us to probe the availability architectural approach and formed the basis for our analysis of this approach, as we shall show below.

It is important to note that by itself Figure 12 (or any similar representation of a software architecture view) tells us little about the system. However, when illuminated by a small number of scenarios and quality attribute-based analysis questions, a view can become the focus for an in-depth understanding and analysis.

Recall that a sensitivity point for a quality attribute is defined as a parameter in the architecture to which some measurable attribute is highly correlated; small changes in such parameters are likely to have significant effects on the measurable behavior of the system. For this reason we need to focus our attention on these points as they pose the highest risks to the system's success, particularly as the system evolves. We find sensitivity points by first using architectural approaches and their associated questions to guide us in our architectural elicitation. Based upon the elicited information we can begin to build models of some aspect of a quality attribute. For example, we might build a collection of formal analytic models such as rate monotonic analysis or queuing models to understand a performance goal such as predicting the worst-case inter-node latency [Klein 93]. We might use Markov models to understand an availability goal such as the expected availability of a system in the face of a server failure [Iannino 94]. We might also employ less formal qualitative analysis models such as that used in SAAM for modifiability [Kazman 94].

These models are frequently quite simple when we initially construct them; often we only have time for a back-of-the-envelope analysis during the 3 days of an ATAM. But when we build even these simple models we can experiment with their parameters until we determine which ones have substantial effects on a measurable attribute such as latency or throughput or mean time to failure. The point of the model building is thus twofold: to find sensitivity points (not to precisely characterize a measurable attribute, for it is typically too early in the development process to do this with any rigor); and to gain insight into the architecture, via the elicitation process that model building necessitates.

For the BCS system we realized, via the utility tree construction process, that three quality attributes were the major architectural drivers for overall system quality: availability, modifiability, and performance. Hence we can say that system quality (Q_S), is a function f of the quality of the modifiability (Q_M), the availability (Q_A), and the performance (Q_P):

$$Q_S = f(Q_M, Q_A, Q_P)$$

The next sections will describe the analyses that we created for performance and availability (the modifiability analysis was simply a SAAM analysis and will not be presented here [Kazman 94]). Each of these analyses was created by first eliciting an architectural approach and then applying quality attribute-specific analysis questions to elicit information about the attribute in question. Using this information we then built simple but adequate analytic models.

10.2.5 Availability

A key quality attribute for the BCS was determined to be its steady-state availability, i.e.,

$$Q_A = g(\text{the fraction of time that the system is working})$$

We determined that the system is considered to be working if there is a working Commander and any number of Soldier nodes. When the Commander fails the system has failed. Provisions had been made in the BCS architecture, however, to turn a Soldier into a Commander, i.e., converting a client into a server. The repair time for the system is the time to turn a Soldier node into the Commander and thus restore the system to operation. Failure of the Commander is detected via human-to-human communication.

As guided by the utility tree entry for availability, the key stimulus to model for the system is the failure of a node (specifically a Commander node) in the system due to an attack, hardware failure, or software failure. The architecture of the BCS, as presented to us, statically specified a Commander and a single backup selected from among the Soldier nodes, as indicated by the shaded Soldier node in Figure 12. In the existing design, *acknowledged* communication (state messages) takes place between the Commander and the backup Soldier node to allow the backup to mirror the Commander's state. This allows the backup to maintain a state of readiness in case of failure of the Commander. Upon failure of the Commander, the backup takes over as the new Commander, converting from a client to a server.

However, there is no provision to have one of the surviving Soldier nodes promoted to become the current backup. So we can refine our characterization of the system's availability as follows:

$$Q_A = g(\lambda_C, \mu_C, \mu_B)$$

That is, system availability is primarily affected by the failure rate of the Commander (λ_C), the repair rate of the Commander (μ_C, the time required to turn the backup into the Commander). The availability might also be affected by the repair rate of the backup, but in the BCS as it is currently designed, the repair rate of the backup (μ_B) is 0 since there is no provision for creating additional backups.

We determined that this is a potential risk. We can understand the nature of this risk by building a simple model of the system's availability. Using this model we can determine the effects of changing the repair rate of the backup on the system's overall availability. Specifically, we can determine the amount of time required for a new backup to enter a readiness state (i.e., where it could quickly become the Commander). This would, however, require a change to the architecture since the current architecture statically specifies only a single backup.

An alternative architecture could allow multiple (perhaps all) Soldier nodes to monitor the Commander-to-backup communication and thus maintain a higher level of readiness. And these additional backups could either acknowledge communication with the Commander (requesting resends of missed packets) or could be silent receivers of packets, or some mixture of these schemes (i.e., the top *n* backups acknowledge receipt of packets, and the remaining *m* backups are passive). In the case where packets are not acknowledged, the state of the backups database would increasingly drift from that of the Commander and if one of these backups is called upon to become the Commander, it would need to engage in some negotiation (with the external systems and/or the other Soldier nodes) to complete its database.

Thus, there are three considerations for changing the BCS architecture to improve the data distribution to the backups:

- a backup could be an "acknowledging backup", which is kept completely synchronized with the Commander
- a backup might be only a "passive" backup and not ask for re-sends when it misses a message; this implies that it has the means for determining that it has missed a message (such as a message numbering scheme)
- a backup, when it becomes the new Commander, or when it becomes an "acknowledging backup," could request any missed information from the upper level Command and Control systems and/or the other Soldier nodes.

The system does not need to choose a single option for its backups. It might have *n* acknowledging backups and *m* passive backups. Assuming that we have no control over the failure rate of the Commander, then the true sensitivities in this system with respect to availability are functions of the repair rates of the Commander and backups, which are themselves functions of the numbers of acknowledging and passive backups. Now we have a usable description of

the architectural sensitivities—a correlation between some architectural parameter and a measurable attribute:

$$Q_A = g(n, m)$$

What are the issues in the choice of the number of backups to maintain and whether they acknowledge communications or not? We consider this issue in terms of the failure and recovery rates of the system under the various options. Clearly, the availability of the system increases as the number of backups is increased, because the system can survive multiple failures of individual nodes without failing its mission. The availability of the system is also increased by increasing the number of acknowledging backups, for two reasons: 1) acknowledging backups can be ready to assume the responsibilities of an Commander much more quickly, because they do not need to negotiate with other nodes for missed information, and; 2) having more acknowledging backups means that there will not be an identifiable communication pattern between the Commander and the single backup, as there is currently, which means that the probability of two accurate incoming mortars disabling the system is reduced.

However, as the number of acknowledging backups is increased, the performance of the system is impacted, as each of these acknowledgments incurs a small communication overhead. Collectively, this overhead is significant, because as we will discuss next, communication latency is the major contributor to overall system latency for BCS.

10.2.6 Performance

Through the building of a simple performance model of the system and varying the input parameters to the model (the various processing times and communication latencies), it became clear that the slow speed of radio modem communication between the Commander and the Soldiers (9600 baud) was the single most important performance driver for the BCS, swamping all other considerations. The performance measure of interest—average latency of client-server communications—was found to be insensitive to all other architectural performance parameters (e.g., the time for the system to update its database, or to send a message internally to a process, or to do targeting calculations). But the choice of modem speed was given as a constraint, and so our performance model was then focused on capturing those architectural decisions that affected message sizes and distributions. Note here that we did not, and did not need to, build a complex RMA or queueing model to discover this correlation.

We thus began our investigation into message sizes by considering the performance implications and the communication requirements implied of the system's normal operations. We then

looked at the system's operations under the assumption of some growth in the performance requirements. To do this we considered three situations when building our performance model:

A) Regular, periodic data updates to the Commander) various message sizes and frequencies.

B) Turning a Soldier node into a backup: a switchover requires that the backup acquires information about all missions, updates to the environmental database, issued orders, current Soldier locations and status, and detailed inventories from the Soldiers.

C) Doubling number of weapons or the number of missions.

We created performance models of each of these situations. For the purposes of illustration in this example, we will only present the performance calculations for situation B), the conversion from Soldier backup to Commander.

After determining that a switchover is to take place the Soldier backup will need to download the current mission plans and environmental database from the external command and control systems. In addition, the backup needs the current locations and status of all of the remaining Soldiers, inventory status from the Soldiers, and the complete set of issued orders.

A typical calculation of the performance implications of this situation must take into account the various message sizes needed to realize the activities, the 9600 baud modem rate (which we equate to 9600 bits/second), and the fact that there are a maximum of 25 Soldiers per Commander (but since one is now being used as a Commander, the number of Soldier nodes in these calculations is 24):

Downloading mission plans:
280 Kbits / 9.6 Kbits/second ≅ 29.17 seconds

Updates to environmental database:
66 Kbits / 9.6 Kbits/second ≅ 6.88 seconds

Acquiring issued orders:
24 Soldiers * (18 Kbits/9.6 Kbits/second) = 45.0 seconds

Acquiring Soldier locations and status:
24 Soldiers * (12 Kbits/9.6 Kbits/second) = 30.0 seconds

Acquiring inventories:
24 Soldiers * (42 Kbits/9.6 Kbits/second) = 105.0 seconds

Total ≅ 216.05 seconds for Soldier to become backup

Note that since the radio modem is a shared communication channel, no other communication can take place while a Soldier/backup is being converted to a Commander.

There was no explicit requirement placed on the time to switch from a Commander to a backup. However, there was an initialization requirement of 300 seconds which we will use in lieu of an explicit switchover time budget. If we assume that the 280K bits in the mission plan file contains the three missions in the current configuration, then doubling the number of missions would imply doubling the mission message from 280K bits to 560K bits and the transmission time would increase by almost 30 seconds, still meeting the time budget. If, on the other hand, the number of Soldiers increases to 35, the total time will increase by about 90 seconds, which would not meet the time budget (when added to the 216.05 seconds for the soldier to become a backup).

Keeping each backup in a state of high readiness requires that they become acknowledging backups, or for a lower state of readiness they can be kept as passive backups. Both classes of backups require periodic updates from the Commander. From an analysis of situation B), we have calculated that these messages average 59,800 Kbits every 10 minutes. Thus, to keep each backup apprised of the state of the Commander requires 99.67 bits/second, or approximately 1% of the system's overall communication bandwidth. Acknowledgments and resends for lost packets would add to this overhead. Given this insight, we can characterize the system's performance sensitivities as follows:

$$Q_P = h(n, m, CO)$$

That is, the system is sensitive to the number of acknowledging backups (n), passive backups (m), and other communication overhead (CO). The main point of this simple analysis is to realize that the size and number of messages to be transmitted over the 9600 baud radio modem is important with respect to system performance and hence availability. Small changes in message sizes or frequencies can cause significant changes to the overall throughput of the system. These changes in message sizes may come from changes imposed upon the system from the outside.

10.2.7 Tradeoff Identification

As a result of these analyses we identified three sensitivities in the BCS system. Two of these are affected by the same architectural parameter: the amount of message traffic that passes over the shared communication channel employed by the radio modems, as described by some functions of n and m, the numbers of acknowledging and passive backups. Recall that availability and performance were characterized as

$$Q_A = g(n, m)$$

and

$$Q_P = h(n, m, CO)$$

These two parameters, *n* and *m*, control the tradeoff point between the overall performance of the system, in terms of the latency over its critical communication resource, and between the availability of the system in terms of the number of backups to the Commander, the way that the state of those backups is maintained, and the negotiation that a backup needs to do to convert to a Commander. To determine the criticality of the tradeoff more precisely, we can prototype or estimate the currently anticipated message traffic and the anticipated increase in message traffic due to acknowledgments of communications to the backups. In addition, we would need to estimate the lag for the switchover from Soldier to Commander introduced by not having acknowledged communication to the Solder backup nodes. Finally, all of this increased communication needs to be considered in light of the performance scalability of the system (since communication bandwidth is the limiting factor here).

One way to mitigate against the communication bandwidth limitation is to plan for new modem hardware with increased communication speeds. Presumably this means introducing some form of indirection into the modem communications software—such as an abstraction layer for the communications—if this does not already exist. This modifiability scenario was not probed during the evaluation.

While this tradeoff might seem obvious, given the presentation here, it was not so. The contractor was not aware of the performance and availability implications of the architectural decisions that had been made. In fact, in our initial pre-meeting, not a single performance or availability scenario was generated by the contractor; these simply were not among their concerns. The contractor was most worried about the modifiability of the system, in terms of the many changes in message formats that they expected to withstand over the BCS's lifetime. However, the identified tradeoff affected the very viability of the system. If this tradeoff was not carefully reasoned, it would affect the system's ability to meet its most fundamental requirements.

10.2.8 Scenario-Based Analysis

Scenarios represent uses of—stimuli to—the BCS architecture. These are applied not only to determine if the architecture meets a functional requirement, but also for further understanding of the system's architectural approaches and the ways in which these approaches meet the quality requirements such as performance, availability, modifiability, and so forth.

The scenarios for the BMS ATAM were collected by a round-robin brainstorming activity in which no criticism and little or no clarification was provided. Table 3 shows a few of the 40 growth-and-exploratory scenarios that were elicited in the course of the ATAM evaluation.[1]

Scenario	Scenario Description
1	Same information presented to user, but different presentation (location, fonts, sizes, colors, etc.).
2	Additional data requested to be presented to user.
3	User requests a change of dialog.
4	A new device is added to the network, e.g., a location device that returns accurate GPS data.
5	An existing device adds additional fields that are not currently handled to existing messages.
6	Map data format changes.
7	The time budget for initialization is reduced from 5 minutes to 90 seconds.
8	Modem baud rate is increased by a factor of 4.
9	Operating system changes to Solaris.
10	Operating schedule is unpredictable.
11	Can a new schedule be accommodated by the OS?
12	Change the number of Soldier nodes from 25 to 35.
13	Change the number of simultaneous missions from 3 to 6.
14	A node converts from being a Soldier/client to become a Commander/server.
15	Incoming message format changes.

Table 3: Examples of Scenarios for the BCS Evaluation

10.2.9 Scenario Prioritization

Prioritization of the scenarios allows the most important scenarios to be addressed within the limited amount of time (and energy) available for the evaluation. Here, "important" is defined entirely by the stakeholders. The prioritization was accomplished by giving each stakeholder a fixed number of votes; 30% of the total number of scenarios has been determined to be a useful heuristic. Thus, for the BCS, each stakeholder was given 12 votes that they use to vote for scenarios in which they were most interested. Typically, the resulting totals will provide an obvious cutoff point; 10-15 scenarios are the most that can be considered in a normal one-day session; for the BCS a natural cutoff occurred at 12 scenarios. Some negotiation is appropriate

1. These scenarios have been cleansed of proprietary specifics, but their spirit is true to the original.

in choosing which scenarios to consider; a stakeholder with a strong interest in a particular scenario can argue for its inclusion, even if it did not receive a large number of votes in the initial prioritization.

10.2.10 Analyze Architectural Approaches

In the ATAM process, once a set of scenarios have been chosen for consideration, these are "mapped" onto the architecture as test cases of the architectural approaches that have been documented. In the case of a scenario that implies a change to the architecture, the architect demonstrates how the scenario would affect the architecture in terms of the changed, added, or deleted components, connectors, and interfaces. For the use case scenarios, the architect traces the execution path through the relevant architectural views.

Stakeholder discussion is important here to elaborate the intended meaning of a scenario description and to discuss how the mapping is or is not suitable from their perspective. The mapping process also illustrates weaknesses in the architecture and its documentation, or even missing architectural approaches.

For the BCS, each of the high-priority scenarios were mapped onto the appropriate architectural approach. For example, when a scenario implied a modification to the architecture, the ramifications of the change were mapped onto the source view, and scenario interactions were identified as sensitivity points. For availability and performance, use case scenarios describing the execution and failure modes of the system were mapped onto run-time and system views of the architecture. During this mapping the models of latency and availability that we had built in the style-based analysis phase were further probed and refined.

11 Results Of The BCS ATAM

The ATAM that we performed on the architecture for the BCS revealed some potentially serious problems in the documentation of the architecture, the clarity of its requirements, its performance, its availability, and a potential architectural tradeoff. We will briefly summarize each of these problems in turn.

11.1 Documentation

The documentation provided at the inception of this project was minimal: two pages of diagrams that did not correspond to software artifacts in any rigorous way. This is, in our experience, typical, and is the single greatest impediment to having a productive architecture evaluation. Having a distinct Phase 1 and having the opportunity to request additional documentation from the contractor made the evaluation successful.

As a result of our interaction with the BCS contractor team, substantially augmented and higher quality architectural documentation was produced. This improved documentation became the basis for the evaluation. And the improvement in the documentation was identified by management as a major success of the ATAM process, even before we presented any findings.

11.2 Requirements

One benefit of doing any architectural evaluation is increased stakeholder communication, resulting in better understanding of requirements. Frequently, new requirements surface as a result of the evaluation. The BCS experience was typical, even though the requirements for this system were "frozen" and had been made public for over two years.

For example, in the BCS the only performance timing requirements were that the system be ready to operate in five minutes from power-on. In particular, there were no timing requirements for other specific operations of the system, such as responding to a particular order, or updating the environment database. These were identified as lacking by the questions we asked in building the performance model.

Furthermore, there was no explicit switchover requirement, i.e., the time that it takes for a Soldier to turn itself into a Commander is not identified as a requirement. This requirement surfaced as a result of building the availability model.

In addition, there was no stated availability requirement. Two well aimed hits, or two specific hardware failures and the system, as it is currently designed, is out of commission. By the end of the ATAM the stakeholders viewed this as a major oversight in the system's design.

11.2.1 Sensitivities and Tradeoffs

The most important tradeoff identified for the BCS was the communications load on the system, as it was affected by various information exchange requirements and availability schemes. The overall performance and availability of the system is highly sensitive to the latency of the (limited and shared) communications channel, as controlled by the parameters n and m. Not only should the current performance characteristics be modeled, but also the anticipated performance changes in the future as the system scales in its size and scope.

11.3 Architectural Risks

In addition to the sensitivities and tradeoffs, we discovered, in building the models of the BCS's availability and performance, a serious architectural weakness that had not been previously identified: there exists the possibility of an opposing force identifying the distinctive communication pattern between the Commander and the backup and thus targeting those nodes specifically. The Commander and backup exchange far more data than any other nodes in the system. This identification can be easily done by an attacker who could discern the distinctive pattern of communication between the Commander and (single) backup, even without being able to decrypt the actual contents of the messages. Thus, it must be assumed that the probability of failure for the Commander and its backup increases over the duration of a mission under the existing BCS architecture.

This was a major architectural flaw that was only revealed *because* we were examining the architecture from the perspective of multiple quality attributes simultaneously. This flaw is, however, easily mitigated by assigning multiple backups, which would eliminate the distinctive communication pattern.

11.4 Reporting Back and Mitigation Strategies

As described above, after the evaluation the contractors were sent a detailed report of the ATAM. In particular, they were alerted to the potential modifiability, performance, and availability problems in the BCS. As an aid to risk mitigation, the contractor was furnished with three architectural possibilities for adding multiple Soldier backups and keeping them more or

less synchronized with the Commander. Each of these architectural changes had their own ramifications that needed to be modeled and assessed. Each of these alternatives will respond differently depending on other architectural choices that affect the performance model, such as the radio modem speed.

The point of this example is not to show which alternative the contractor chose, for that is relatively unimportant and relied on their organizational and mission-specific constraints. The point here is to show that the process of performing an ATAM on the BCS raised the stakeholders' awareness of critical risk, sensitivity, and tradeoff issues. This, in turn, focused design activity in the areas of highest risk and caused a major iteration within the spiral process of design and analysis.

12 Conclusions

An architecture analysis method, any architecture analysis method, is a garbage-in-garbage-out process. The ATAM is no different. It crucially relies on the active and willing participation of the stakeholders (particularly the architecture team); some advance preparation by the key stakeholders; an understanding of architectural design issues and analytic models; and a clearly articulated set of quality attribute requirements and a set of business goals from which they are derived. Our purpose in creating a *method* (rather than, say, just putting some intelligent and experienced people together in a room and having them chat about the architecture or inspect it in an arbitrary way) is to increase the effectiveness and repeatability of the analysis.

To this end we have not only defined a set of steps and their associated stakeholders, but have also amassed a set of techniques, guidance, documentation templates, standard letters and agendas, and so forth. We have created quality attribute characterizations and sets of associated questions. We have documented attribute-based architectural styles. We have created sample templates for the evaluators to fill in with architectural information. We have developed guidelines and techniques for eliciting and voting on scenarios. We created the idea of a utility tree, along with ways of both eliciting and prioritizing it. The ATAM is an infrastructure that puts this set of techniques and artifacts together into a working repeatable whole.

The architectures of substantial software-intensive systems are large and complex. They involve many stakeholders, each of whom has their own agenda. These architectures are frequently incompletely thought out or only partially documented. A method for architecture evaluation has to consider and overcome all of these daunting challenges. Our techniques are aimed at taming this considerable complexity and ensuring the achievement of the main goals for an ATAM: to obtain a precise statement of the quality attribute requirements; to understand the architectural decisions that have been made; and to determine if the architectural decisions made adequately address the quality attribute requirements.

Bibliography

[Bass 98] Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice*. Reading, MA: Addison-Wesley, 1998.

[Buschmann 96] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; & Stal, M. *Pattern-Oriented Software Architecture*. Chichester, UK: Wiley, 1996.

[Iannino 94] Iannino, A. "Software Reliability Theory." *Encyclopedia of Software Engineering,* New York, NY: Wiley, May 1994, 1237-1253.

[Kazman 99] Kazman, R. & Carriere, S.J. "Playing Detective: Reconstructing Software Architecture from Available Evidence." *Automated Software Engineering, 6*, 2 (April 1999): 107-138.

[Kazman 94] Kazman, R.; Abowd, G.; Bass, L.; & Webb, M. "SAAM: A Method for Analyzing the Properties of Software Architectures," 81-90. *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy, May 1994.

[Klein 99] Klein, M. & Kazman, R. *Attribute-Based Architectural Styles* (CMU/SEI-99-TR-022, ADA371802). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. Available WWW. URL: <http://www.sei.cmu.edu/publications/documents/99.reports/99tr022/99tr022abstract.html>.

[Klein 93] Klein, M.; Ralya, T.; Pollak, B.; Obenza, R.; & Gonzales Harbour, M. *A Practitioner's Handbook for Real-Time Analysis*. New York, NY: Kluwer Academic, 1993.

[Leung 82] Leung, J.L.T. & Whitehead, J. "On the Complexity of Fixed Priority Scheduling of Periodic, Real-Time Tasks." *Performance Evaluation, 2,* 4 (1982) 237-250.

[Rajkumar 91] Rajkumar, R. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Boston, MA: Kluwer Academic Publishers, 1991.

[Saaty 94] Saaty, T. *Fundamentals of Decision Making and Priority Theory With the Analytic Hierarchy Process*. Pittsburgh, PA: RWS Publications, 1994.

[Shaw 96] Shaw, M. & Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ: Prentice-Hall, 1996.

Appendix A Attribute Characteristics

A.1 Performance

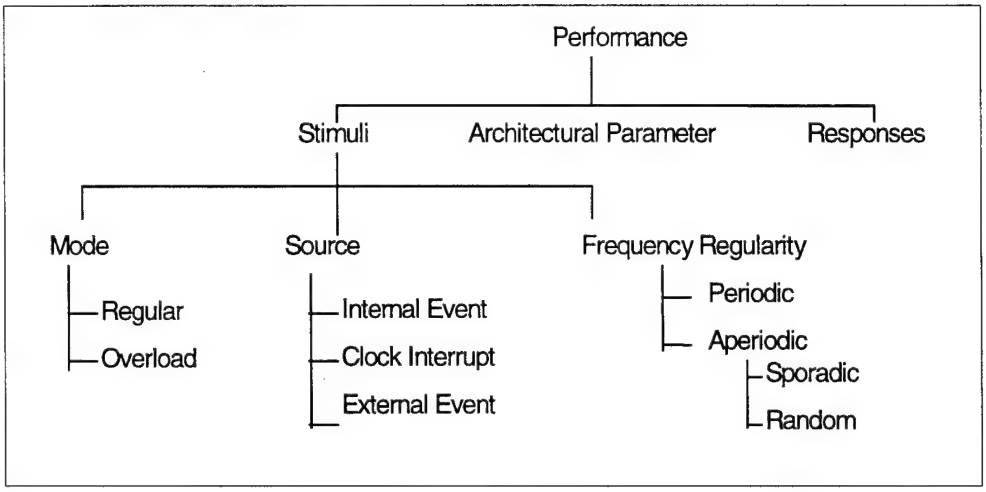

Figure 14: Performance Characterization—Stimuli

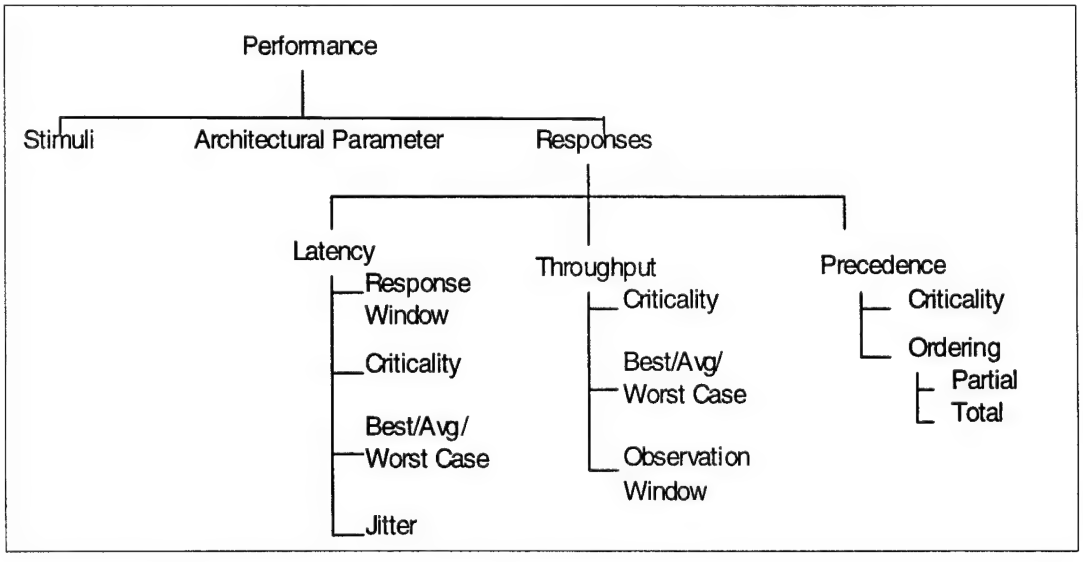

Figure 15: Performance Characterization—Responses

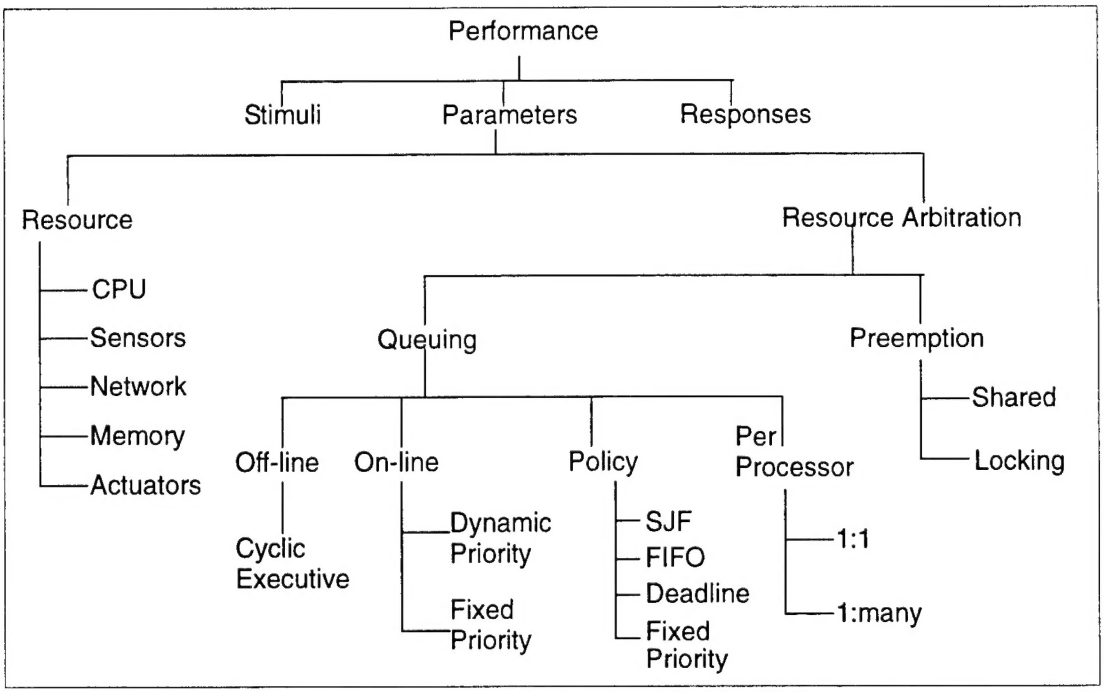

Figure 16: Performance Characterization—Architectural Decisions

A.2 Modifiability

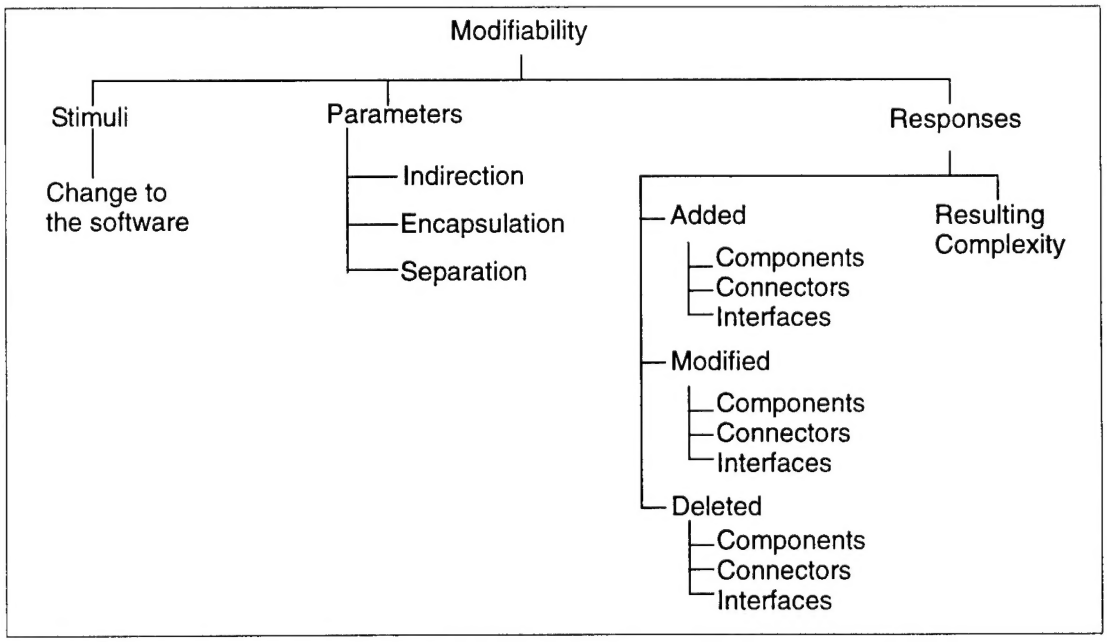

Figure 17: Modifiability Characterization

A.3 Availability

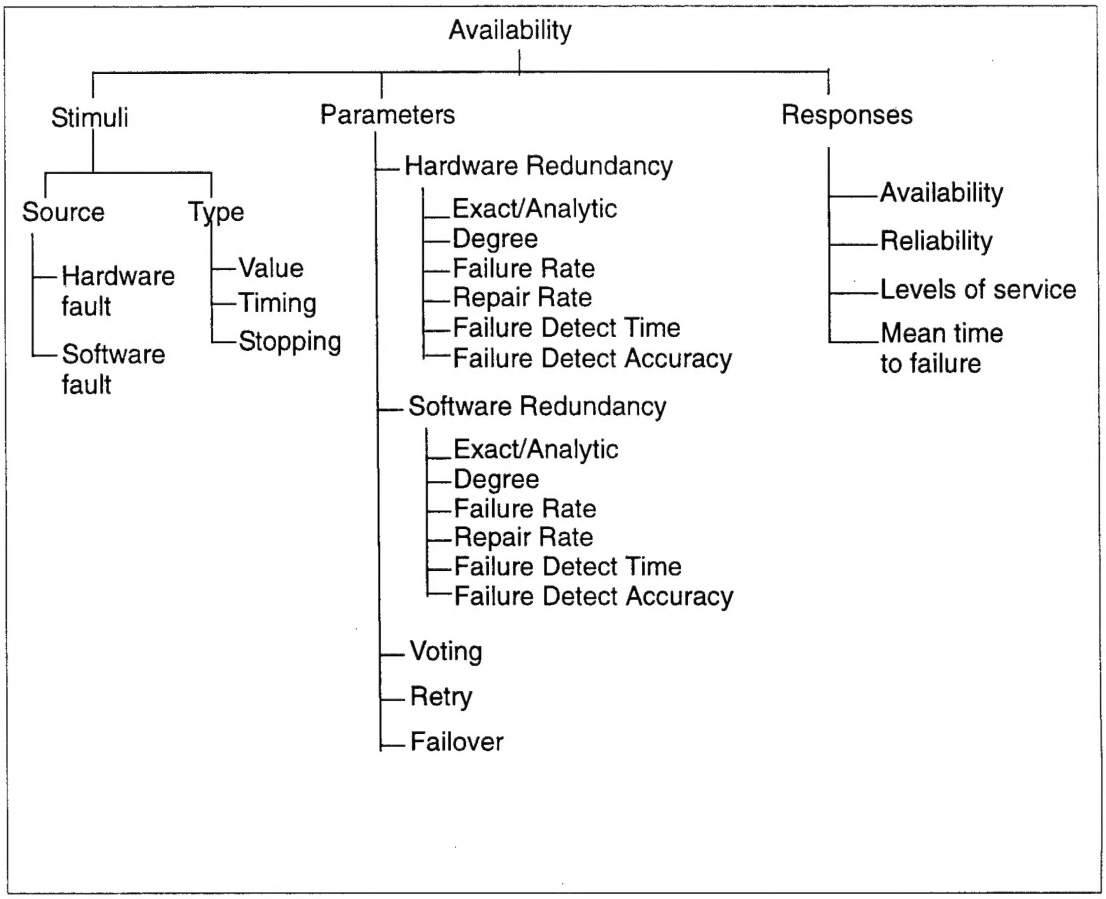

Figure 18: Availability Characterization

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

13. AGENCY USE ONLY (leave blank)	14. REPORT DATE	15. REPORT TYPE AND DATES COVERED
	August 2000	Final

16. TITLE AND SUBTITLE	17. FUNDING NUMBERS
ATAM: Method for Architecture Evaluation	C — F19628-95-C-0003
18. AUTHOR(S)	
Rick Kazman, Mark Klein, Paul Clements	

19. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	20. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2000-TR-004
21. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	22. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2000-004

23. SUPPLEMENTARY NOTES

12.a DISTRIBUTION/AVAILABILITY STATEMENT	12.b DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (maximum 200 words)

If a software architecture is a key business asset for an organization, then architectural analysis must also be a key practice for that organization. Why? Because architectures are complex and involve many design tradeoffs. Without undertaking a formal analysis process, the organization cannot ensure that the architectural decisions made—particularly those which affect the achievement of quality attribute such as performance, availability, security, and modifiability—are advisable ones that appropriately mitigate risks. In this report, we will discuss some of the technical and organizational foundations for performing architectural analysis, and will present the Architecture Tradeoff Analysis MethodSM (ATAM)—a technique for analyzing software architectures that we have developed and refined in practice over the past three years.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software architecture, architectural analysis, ATAM, quality attributes	84
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL